# Middleware Systems and Analytics for Energy Management in Buildings

By
Pandarasamy Arjunan

**Under the supervision of**
Dr. Amarjeet Singh
Dr. Pushpendra Singh

Indraprastha Institute of Information Technology Delhi

March 2018

# Middleware Systems and Analytics for Energy Management in Buildings

by
Pandarasamy Arjunan

Submitted
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

to the

Indraprastha Institute of Information Technology Delhi
New Delhi, India

# Certificate

This is to certify that the thesis titled **Middleware Systems and Analytics for Energy Management in Buildings** being submitted by **Pandarasamy Arjunan** to the Indraprastha Institute of Information Technology Delhi, for the award of the degree of Doctor of Philosophy, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Supervisors

Dr. Amarjeet Singh

Dr. Pushpendra Singh

March 2018

Indraprastha Institute of Information Technology Delhi
New Delhi 110 020

**Keywords:** Smart buildings, Building management system, Middleware system, Energy data analytics, and Anomaly detection

*dedicated to my mother...*

# Abstract

As one of the largest consumers of overall energy, buildings have emerged as attractive targets for using information and communications technologies to advance large-scale sustainability goals. With increasing availability and affordability of sophisticated sensing, control and computational methods, a variety of novel applications have been envisioned in the recent past aiming towards energy savings. However, the centralized building management system with its inflexible and isolated subsystems, currently used to manage the building operations, restrict the widespread development and deployment of novel energy management applications. In this thesis, we hypothesize that decentralized, flexible, and extensible software systems, together with novel applications and analytical techniques, would improve the energy efficiency in buildings.

To support our hypothesis, we present the architecture, design, development, and experimental validation of middleware systems for building energy management, which enable 1) decentralized management of building resources involving different stakeholders, including occupants, to make control-over decisions and energy management policies, by providing appropriate fine-grained access-control mechanisms, 2) flexible interfaces for integrating existing and retrofitted sensing and control systems, and suitable software representations for accessing and managing their operations, e.g. spatio-hierarchical relationship, which are specific to buildings, and 3) an extensible automation framework for developing and deploying energy management applications involving simple and advanced sensor data processing methods for identifying detailed insights about the operational context of the building, and suitable programming abstractions for developers.

We evaluate these systems through multiple real-world deployments in our test-bed buildings consisting of varied categories of functionalities, operations, users, and hundreds of heterogeneous sensor data streams, across the world. On top of this, we also implemented several practical applications ranging from detecting the deviation in the energy usage of building sub-

systems to inferring fine-grained building context using proxy sources. The practical usability of the system was also evaluated through a user study. In summary, this thesis attempts to present a holistic software ecosystem and novel applications by bringing the three major entities – devices, computational methods, and humans, in buildings closer towards optimal energy management.

# Acknowledgments

First of all, I would like to thank Prof. Pankaj Jalote for providing an opportunity for pursuing my PhD at IIIT-Delhi. I am deeply indebted to my advisors, Dr. Amarjeet Singh and Dr. Pushpendra Singh, for their great mentorship while I was working towards the completion of this thesis. I am very grateful for their insights and support during this process and have learned a great deal from them about all facets of research. I am also very thankful to Prof. Mani Srivastava (NESL, UCLA) for his guidance and active collaboration with us as part of the Indo-US PC3 project.

Special thanks to my PhD thesis examiners, Prof. Krithi Ramamritham, Prof. Prashant Shenoy, and Prof. Anand Sivasubramaniam for providing invaluable feedback for improving this thesis. I have also been fortunate to receive quality feedback from the research community, particularly the program committee members of BuildSys, eEnergy, MobiQuitous, among others.

I am very thankful to all the faculty members of Mobile and Ubiquitous Computing group, particularly Dr. Vinayak Nayak, and other faculty members of IIIT-Delhi for providing necessary guidance and support whenever required. I also would like to thank all administrative staffs and facilities department for providing all possible support and help.

I have been grateful to have collaborated and worked with my lab mates Siddhartha Asthana, Nipun Batra, Manoj Gulati, Manaswi Saha, Milan Jain, Haroon, Dheryta, Anil, Garvita, and other PhD students and interns. Thank you guys for providing constructive feedback, comments, motivation, support, and of course, all fun. I am also thankful to Haksoo, Supriyo, Lucas and other lab mates at NESL, UCLA for their guidance and support while my stay in the USA. Special thanks to Sumesh, Jayaprakash, Monalisa, Swetha, Amani, Harish and Prabha for their friendship, moral support, and making my stay in Delhi a memorable one.

I am deeply thankful to Mrs. Umayaparvathi, who made me who I am today. She is the source of motivation and a mentor of my academic carrier. I also would like to thank all my friends from my undergraduate and postgraduate courses and ex-colleagues from HCL for their continuous encouragement throughout my PhD life, particularly Senthilkumar, Marimuthu, Saravanan, Raj, Alex, Esakki, Pandian, Karthick prakash, Brijesh, and Nithya. Special thanks to Christina John who made me join IIIT-Delhi and for her continuous motivation during my early days of PhD.

Last but not least, I am indebted to all my family members, particularly my wife Rajeshwari and my son Hariharan for supporting me all possible ways for completing my PhD thesis.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Smart Energy Buildings

Buildings account for the largest proportion of overall energy use in both developing (e.g. 47% of total energy in India [87]) and developed (e.g. 41% in the USA [60] ) countries. They consume 72% of total electricity and emit approximately 40% of greenhouse gases (GHG) annually in the USA itself [50]. The Sankey diagram in Figure 1.1 shows the energy flow across different sectors within the USA as of 2015 [13]. It also shows how much energy is consumed by residential and commercial buildings and their inefficiencies. Further, rapid urbanization in developing countries is resulting in substantial increase in building floor space with inefficient Energy Performance Index (EPI)[1].

According to the Smart 2020 report [89], buildings have been identified as one of the primary targets to reduce the overall energy consumption to achieve the energy sustainability goals and to reduce the greenhouse gas emissions. With a large number of buildings, even modest improvements will lead to significant impact at the national aggregate scale. As a result, there is tremendous research and entrepreneurial activity targeting buildings as an exciting substrate for

[1]EPI of 200-400 $kWh/m^2/year$ in India while comparable buildings in North America and Europe have $EPI < 150kWh/m^2/year$

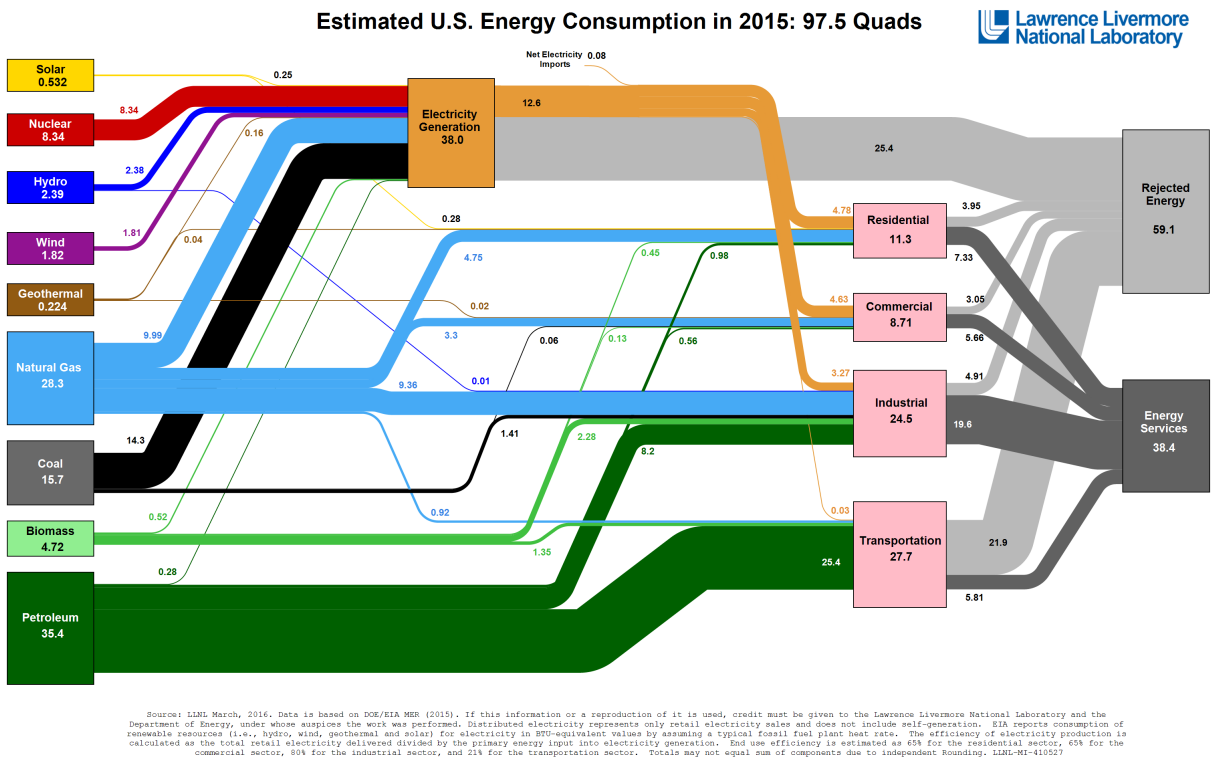
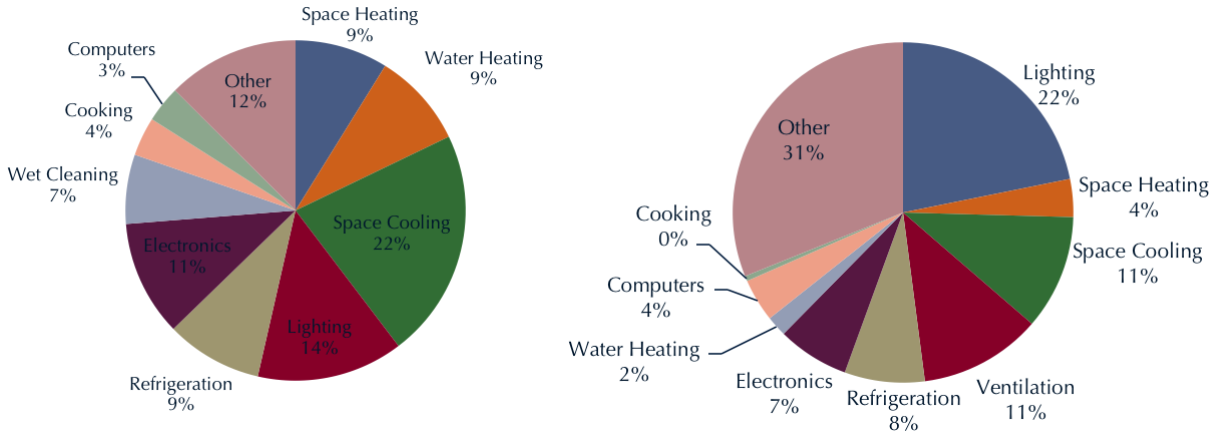



Figure 1.1: The sankey diagram showing the energy flow from sources of production to consumptions across different sectors in the USA for the year 2015 [13]. It is estimated that 35% of energy is wasted in both residential and commercial buildings.

novel computational methods and technologies. These techniques measure, model, analyze, and optimize the design and operations of the buildings.

Electricity consumption in buildings is spread across diverse subsystems such as Heating Ventilation and Air Conditioning (HVAC), lighting, entertainment, security and safety systems, and Miscellaneous Electronic Loads (MELs). Modern buildings are also instrumented with several Information and Communications Technology (ICT) systems like computers, printers, wired and wireless networking devices, which in turn increase the net electricity consumption. As an example, Figure 1.2a and Figure 1.2b illustrate end-use energy consumption of residential and commercial buildings, respectively, in the USA for the year 2010. It also shows that lighting, space heating, cooling and ventilation systems are the topmost energy consumers in both residential and commercial buildings.




(a) Residential buildings

(b) Commercial buildings

Figure 1.2: Electricity consumption of residential [39] and commercial [7] buildings by end-use in the USA for the year 2010.

Several techniques have been proposed in the past decade by both research community and several governments and industry sectors to reduce the energy footprint of buildings. There are two broad categories of techniques found in the literature: *Supply-side management (SSM)* and *Demand-side management (DSM)*. Supply-side management involves improving the existing energy generation systems or adopting renewable energy sources such as solar and wind power [10]. Whereas, demand-side management optimizes individual building's energy budget. DSM is performed using Demand-Response (DR) programs or energy efficiency methods [12]. DR programs motivate the consumers to reduce energy consumption during peak hours or shift some loads to off-peak hours, thus reduce the demand-supply mismatch.

Energy efficiency methods are targeted towards optimizing the overall energy usage within buildings, resulting in the permanent reduction of energy demand. They are primarily targeted towards manipulating the three major elements: 1) building-centric (retrofitting the building's fixtures); 2) human-centric (changing the occupant's energy usage behavior); and 3) Cyber-Physical Systems (CPS) based control systems and applications.

The building-centric retrofitting methods primarily focus on identifying energy inefficient fixtures and replacing them with energy efficient ones. Examples include compliance with building energy efficiency codes and standards such as IECC [21] and ASHRAE [1], energy audits [11], blower door testing [5] for detecting thermal leakage in buildings to ensure proper insulation of walls and windows, replacing incandescent bulbs with Compact Fluorescent Lamp (CFL) and Light-emitting diode (LED) lights [6], and replacing inefficient and old refrigerators and air conditioners with energy star rated appliances [14].

Several human-centric methods have also emerged recently focusing on educating the building users about their day-to-day energy usage pattern and subsequently changing their behavior by providing continuous energy usage feedback. Example methods include energy awareness campaigns [41] and games [71], offering a breakdown of electricity bills [44], energy dashboards [62], and In-Home Displays (IHD) [8] showing real-time energy usage. All these methods can help the occupants to take informed decisions based on their everyday electricity usage. This direction is extensively studied by Human-Computer-Interaction (HCI) community [108]. A user study conducted by Darby [81] showed that better feedback systems encourage energy-saving behavior that can save up to 5-15% of the electricity usage. Another study conducted by Bidgely among 850 PG&E customers showed that up to 7.7% of energy savings was achieved when the customers were informed about their energy usage behavior during certain peak hours [3].

CPS-based solutions often involve software and hardware-based systems for energy management in buildings. Examples include occupancy-based lighting [61, 100] and HVAC control [85, 86, 112], automated identification of abnormal energy usage events [67, 88, 114], and identification of HVAC faults [104]. In contrast with building and human-centric solutions, CPS-based solutions make use of both existing and retrofitted sensing and control systems in buildings. Further, they leverage the advanced computational methods for optimizing the energy usage of buildings, thus making them affordable at scale. The primary focus of this thesis is proposing novel software systems and applications for monitoring and reducing the energy usage in buildings.

## 1.2 Building Management Systems

Modern buildings are increasingly instrumented with a large number of networked sensing and control systems such as HVAC, lighting, security, and safety. Building managers often employ individual software systems for managing and automating various building operations. These software systems are commonly referred as Building Management System (BMS). Several commercial systems are currently available in the market for monitoring, controlling and automating various building subsystems and operations. Examples of commercially available BMS systems include Trane [45], Johnson Controls [23], Siemens [43], and Honeywell [20]. These systems comprise of several isolated subsystems, each performing a particular task.

While BMS includes a large number of sensing points spread across the building, the sensor data collected by these subsystems are often discarded after they are processed by their corresponding isolated control applications. Since these sensor data encompass several operational characteristics of a building (e.g., occupancy information, historical energy usage patterns), storing and thereafter analyzing them can lead to identifying potential energy saving measures. As an example, sensor data for temperature and $CO_2$ collected in HVAC systems can be analyzed to extract occupancy information, which in turn can be used for conditioning the devices in unoccupied regions of a building.

With increasing availability and affordability of sophisticated sensing, control and computational methods, a variety of novel energy saving applications have been proposed in the recent past by leveraging the building sensor data. Example applications include occupancy-based lighting, heating and cooling control [61, 86, 101]; energy forecasting for demand-side load management [118]; energy disaggregation for providing appliance level consumption feedback [65, 93]; and automated anomaly detection methods for reducing energy wastage [68, 88, 114]. However, all these efforts in building-scale energy management are fraught with deployment challenges, many of which are rooted in the centralized, inflexible, isolated, and archaic architecture that forms the middleware in the current building management systems.

To cater to these requirements we envision a decentralized energy management system architecture for easy development and deployment of novel energy monitoring applications. Such systems should provide flexible interfaces for integrating and managing the existing heterogeneous sensing and control systems in buildings. It should be extensible for implementing sophisticated application logic using the right abstractions, and scalable for a large number of buildings. In this thesis, we present the design, development, deployment, and validation of novel software systems and analytical techniques for optimizing the energy usage in buildings.

## 1.3 Thesis Contributions

The major contributions of this thesis are twofold: First, we present the design, development, and evaluation of a suite of software systems and novel applications for optimal energy management in buildings. The proposed software systems provide several key functionalities such as, (1) decentralized and distributed management of building resources facilitating occupant-level energy management policies, (2) a flexible system for integrating the existing and retrofitted sensing and control systems and their data into a single platform for better resource management, (3) a scalable service for sensor data analytics to infer the operational context of the building, and (4) an extensible automation framework for implementing novel energy monitoring applications.

Secondly, we present an unsupervised and scalable analytical technique for identifying aberrations for a network of buildings within a neighborhood, using only the aggregate-level smart meter data. Prototype implementations of the proposed software systems and methods for improving the energy efficiency of buildings were deployed in our testbed buildings. In addition to this, a number of practical energy monitoring applications are developed to validate how the proposed systems are helpful in identifying the real-world energy wastage events. A summary of the proposed systems and their contributions are described below.

The first system that we describe is a decentralized, flexible, and extensible middleware system architecture, called *SensorAct*, for energy management in buildings. In addition to providing

support for managing and integrating heterogeneous sensing and actuation systems in buildings, *SensorAct* architecture provides two novel features: 1) a scripting framework for extending and automating the various energy management functions of the modern buildings, and 2) a rule-based fine-grained mechanism for sharing sensor data and control across building applications and stakeholders, including occupants. The *SensorAct* system provides abstractions, through RESTful APIs, for accessing the underlying networked resources which enable the development of extensible energy monitoring applications. We describe the system design in detail and provide a proof of concept through multiple third-party applications built using *SensorAct* APIs, and deployment in diverse settings across India and United States. We developed several energy monitoring applications using the *SensorAct* system, such as 1) monitoring and detecting aberrations in the street lighting system, 2) irregular usage of the sports area lighting system, and 3) campus-wide critical energy usage alerts.

The second system is focused on providing sensor data analytics support for identifying the fine-grained operational context of the buildings. The scripting framework in *SensorAct* provides support for detecting aberrations by applying threshold based conditions over raw sensor data readings. Though it is simple and powerful for detecting aberrations, it is insufficient in detecting complex context based aberrations, which require advanced sensor data analytics for identifying the operational context of the buildings e.g. operating state of a device, and occupancy levels of a building region. We presented *OpenBAN*, a middleware system service which provides a framework of extensible sensor data processing elements for identifying various building context using historical and real-time sensor data. The computed context information can be shared across multiple energy monitoring applications.

*OpenBAN* architecture was designed to scale across local and cloud-based deployments and to support a 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* was

designed to enable building facilities department to connect various sensor data streams with different existing and new control applications through a powerful analytics engine capable of inferring context information. 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) identifying occupancy information using network activity. Both *SensorAct* and *OpenBAN* are released in open-source for the community use.

Finally, we present a novel unsupervised method for monitoring and identifying energy usage aberrations for a large number of buildings within a neighborhood using aggregate level smart meter data collected by utilities. The proposed approach recognizes that sensing every possible context variable (such as occupancy, internal and external temperature), which affects the energy usage, is technically and economically infeasible. Instead, it uses the context information which is directly available from meter readings: temporal context and metadata attached to the meter identity. Temporal context information, such as operating hours and seasonal changes, is used for improving the aberration detection accuracy by picking only the relevant historical data for baseline estimation. Neighborhood information (as derived from available metadata) is used for adjusting the aberration score to account for unknown context variables (e.g. local festivals and holidays) that influence historically correlated consumption patterns in the same way. We validated the effectiveness of the proposed method using real-world smart meter readings for both commercial and residential buildings. We showed that using context information improves the detection accuracy, and it outperforms a baseline method proposed in the literature.

In summary, the software systems presented in this thesis provide platform support for the development and deployment of extensible building applications pertaining to the automated identification of energy usage aberrations in the building subsystems.

## 1.4 Thesis outline

The structure of the remainder of this thesis is as follows:

In Chapter 2, we describe the design, development, deployment and experimental validation of the *SensorAct* system. We also present the details of a candidate set applications for detecting real-world abnormal energy usage events in our test-bed buildings

In Chapter 3, we describe the design, development, and experimental validation of *OpenBAN* system for sensor data analytics. We also present the requirements and design considerations for making it extensible and scalable. Finally, we show how the *SensorAct* and *OpenBAN* systems can be integrated for developing novel energy monitoring applications.

In Chapter 4, we describe our unsupervised aberration detection method for detecting abnormal energy usage for a network of buildings within a neighborhood. We explain how the readily available metadata can be used as context variables for improving aberration detection method accuracy.

In Chapter 5, we conclude this thesis with a summary of contributions, limitations and outline the future directions.

## 1.5 Thesis publications

- Our work in Chapter 2 was published in the following proceedings:

  Arjunan, Pandarasamy, Nipun Batra, Haksoo Choi, Amarjeet Singh, Pushpendra Singh, and Mani 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. Arjunan,

  Pandarasamy, Manaswi Saha, Haksoo Choi, Manoj Gulati, Amarjeet Singh, Pushpendra Singh, and Mani B. Srivastava. "SensorAct: A Decentralized and Scriptable Middleware for Smart Energy Buildings." In Proceedings of the 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), pp. 11-19. IEEE, 2015.

- Our work in Chapter 3 was published as:

  Arjunan, Pandarasamy, Mani Srivastava, Amarjeet Singh, and Pushpendra Singh. "OpenBAN: An Open Building ANalytics Middleware for Smart Buildings." In proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, pp. 70-79. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015.

- Our work in Chapter 4 was published as:

  Arjunan, Pandarasamy, Harshad D. Khadilkar, Tanuja Ganu, Zainul M. Charbiwala, Amarjeet Singh, and Pushpendra Singh. "Multi-user energy consumption monitoring and anomaly detection with partial context information." In Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments, pp. 35-44. ACM, 2015.

# Chapter 2

# *SensorAct* – A Decentralized Energy Management System for Buildings

The archaic centralized building management systems, currently used to manage buildings, make it hard to incorporate advances in sensing technology and user-level applications. They are installed with a fixed number of isolated and closed-loop control applications which are not extensible. In this chapter, we present *SensorAct*, a novel software platform that enables the development of extensible applications for monitoring and identifying the energy wastage events in buildings. *SensorAct* provides two novel features, in addition to providing flexible interfaces for integrating heterogeneous sensing and actuation systems in buildings for better management: 1) a *scripting framework* for extending and automating the energy management functions of the modern buildings, and 2) a *rule-based sensor data and control sharing* mechanism for fine-grained sharing of sensor data and control across building applications and various stakeholders. We describe the detailed system design, implementation, and provide proof of concept through several energy monitoring applications built using *SensorAct* APIs. We also show that how the scripting framework is useful in detecting real-world energy usage aberrations by defining simple threshold-based rules.

## 2.1 Background and Motivation

Commercial buildings often employ BMS for monitoring and controlling their subsystems. While striving to achieve an optimal energy efficient control, these BMSes restrict the management to a central facility department. They typically provide a stand-alone package of fixed, pre-configured applications and limited support for control and automation. Their archaic nature limits integration of different subsystems, addition of new sensors, and development of novel energy monitoring and control applications. Their legacy programming models and application-specific jargons making them difficult to extend the system by non-expert users[98].

To support interoperability, many of these BMSes support open standard protocols such as BACnet [2]. However, external third-party application development using BACnet interface is complex; hence it restricts widespread development of applications [98]. Such third party applications, if facilitated, can further support the integration of different building subsystems for improved operations. For example, information from Radio-frequency identification (RFID) based access control system can be used to infer the occupancy and accordingly control the HVAC and lighting systems.

Motivated by these limitations we present *SensorAct*, a middleware system architecture designed for detailed monitoring and control of buildings. Beyond connecting devices and thereby monitoring the built spaces, *SensorAct* provides emerging capabilities including:

1. *Virtual Personal Device Servers (VPDS)* (See Section 2.2.2) for local hosting of middleware within the buildings to alleviate data sharing, control security and intermittent network connectivity problems.
2. Decentralized and distributed management of building resources involving diverse devices and different stakeholders, including the occupants, at scale.
3. Fine-grained selective sharing of sensor data and control with users at the global scale to alleviate control security concerns and providing building-wise local storage for protecting the data privacy.

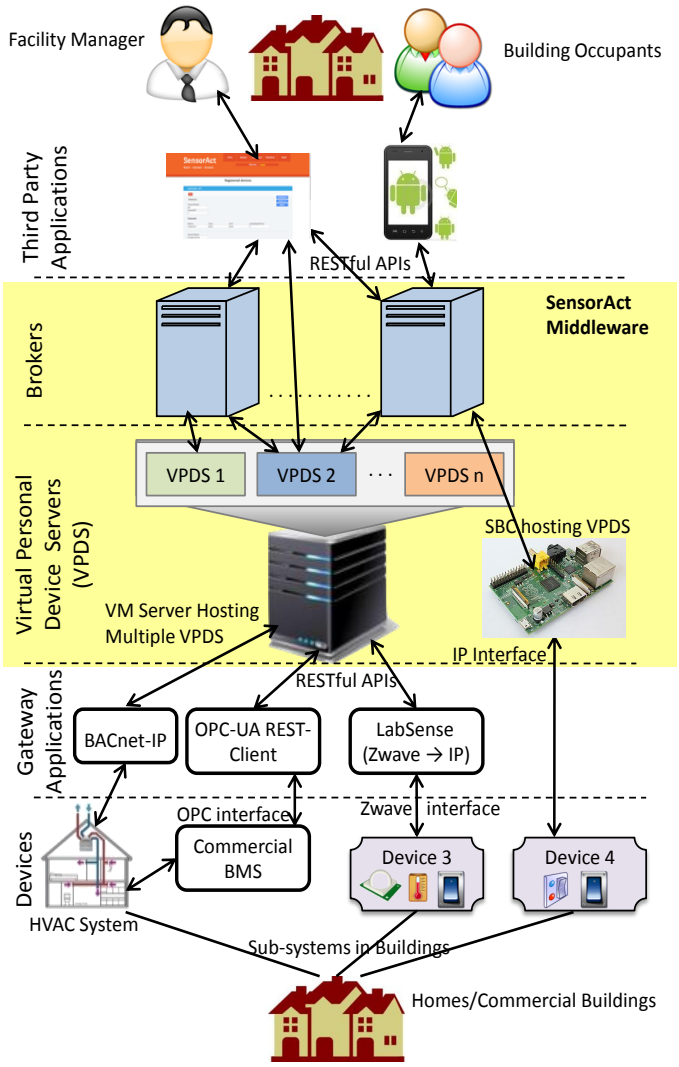



Figure 2.1: *SensorAct* system architecture showing various layers and system components.

4. *Scripting framework* (See Section 2.3), simple programming abstractions for extending the system features and developing energy managing applications involving sensing and control, analysis, alerts, and notifications in rich form to support diverse usage scenarios.

Together with supporting diverse usage scenarios, *SensorAct* accommodates a rich ecosystem of existing and new monitoring and control systems for the development of extensible building applications.

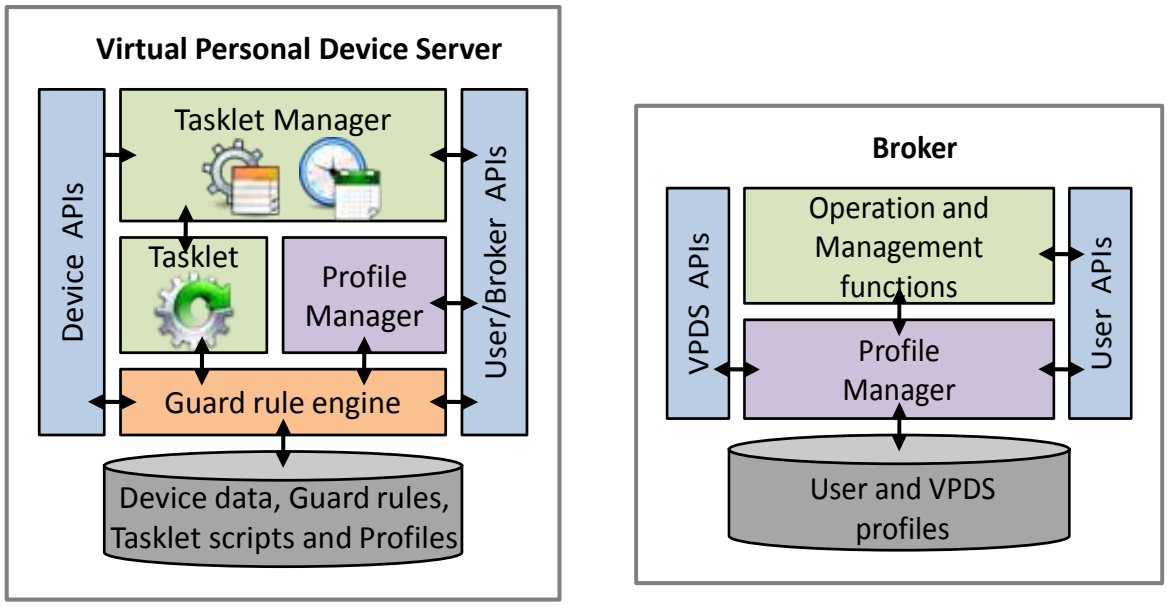
## 2.2 SensorAct System Architecture

*SensorAct* adopts a *tiered architecture* connecting the building subsystems with control applications and occupants as shown in Figure 2.1. The main layers of *SensorAct* are *Virtual Personal Device Servers (VPDS)* and *Brokers*, which interact with *Devices* that monitor and control different building spaces at the bottom layer, and third-party *applications and users* at the top layer.

### 2.2.1 Devices and Gateways

Existing building subsystems contain numerous sensing and actuation points connected using diverse protocols such as BACnet and Modbus [27]. *SensorAct* maps the existing and new sensing and actuation points as *Devices* by eliminating the underlying complex naming conventions used in the legacy building subsystems. A device in *SensorAct* consists of a collection of sensors (monitoring the ambient environment or the state of a building subsystem) and actuators (that allow for changing the state of a building subsystem). Each sensor, in turn, may consist of a collection of channels which measure a particular building phenomenon. *SensorAct* uses an intuitive and hierarchical naming scheme for managing and identifying the devices and their associated sensors/actuators, channels and readings uniquely. As an example, a single reading from a smart energy meter connected with the main panel is identified as $OwnerName/MainPanel/EnergyMeter1/power/ < timestamp > / < value >$.

Devices are assumed to either communicate directly or through a gateway (e.g., LabSense [25], as discussed in Section 2.4.1) that can bridge the sensing interface with *SensorAct* using its RESTful API. Each device is associated with a device profile that contains all meta information. A building owner can manage multiple devices owned by him through the *profile manager* facilitating the creation of device profiles. It consists of a set of key-value paired attributes such as its name, IP address, a collection of sensor and actuator profiles, number of channels, data types and units, location, and placement. Besides, the physical sensors and actuators, *SensorAct* also supports:




(a) VPDS Components

(b) Broker Components

Figure 2.2: Components of the VPDS and Broker in *SensorAct*.

**Computed sensors:** are abstractions that can be calculated based on a pre-defined mathematical function applied to a single or multiple sensor values. For example, a computed sensor can specify average temperature in a room over 10 minutes (calculated using temperature values observed every second) and a computed sensor *InMeeting* can be set based on multiple occupants in the room (using a combination of a motion sensor and microphone sensor).

**Grouped actuators:** are abstractions that can control multiple actuators e.g. power off the room may result in turning off air-conditioner as well as lights. These are similar to "scenes" as specified in several commercial home automation systems.

### 2.2.2 Virtual Personal Device Server (VPDS)

VPDS is the primary core component of *SensorAct* middleware architecture. As a package, VPDS contains (a) *Data Archiver* for storing and retrieving time-series sensor data, (b) *Scripting Framework* for executing the custom building applications, (c) *Guard Rule Engine* for access control, and (d) *Profile Manager* and APIs for devices, applications, and brokers to interact with

the VPDS, as shown in Figure 2.2. To ensure scalability, a single physical server may host multiple VPDS, however virtualization is used to, ensure the isolation of VPDS and privacy of the data within it. A building owner who owns a particular VPDS is called as VPDS Owner (VO) in *SensorAct*. VO may allow other users and applications to have controlled access to the devices registered with the VPDS. These devices monitor and control the buildings owned by VO. Each VPDS has an associated auto-generated owner's key which is used while registering the VPDS with a Broker for data and control sharing. All communication between VPDS and VO is authenticated using the owners key.

A schema-less, key-value based database (described in Section 2.4.2) is used to store the sensor data from devices, allowing for efficient storage and querying of unstructured time-series data. Access to the database is guarded via the *Guard Rule Engine* (see Section 2.2.2) in the VPDS. Lightweight scripts are used to act on configured triggers to support complex automation tasks (see Section 2.3). The VPDS abstraction not only permits flexible provisioning of hardware resources but also allows diverse deployment scenarios including a VPDS hosted on a local machine for high-frequency sensors, low dependence on Internet connectivity, high-security and low-latency sensing and control. Such multiple VPDS instances are coordinated through Brokers in the higher tier. VPDS-Broker architecture further enables global access through the distributed registry of users at scale.

**Sensory Information Representation**

*SensorAct* system needs to handle large volumes of data generated by a multitude of sensors. Storing the time series of sensor data as individual tuples is inefficient both in terms of storage size and querying time. In order to provide an abstraction of sensors that is generic, compact, efficient, and scalable to diverse sensing modalities and sampling policies, *SensorAct* represents the continuous sensor data streams using *WaveSegs*, an abstraction for sensor waveforms used in [79] and in turn inspired by *SigSeg* used in MIT's WaveScript system [105]. WaveSegs refers to

non-overlapping windows of the sensor waveform, and are the atomic units from which a sensor waveform in *SensorAct* is composed. Sensors send measurements to *SensorAct* as WaveSegs, and *SensorAct*'s storage is also organized in terms of WaveSegs instead of individual samples.

Within a WaveSeg the sampling policy is fixed, with support for both isochronous (periodic or uniform) sampling and asynchronous (aperiodic or non-uniform or adaptive) sampling. In the former case, *SensorAct* leverages isochronicity for compactness of representation by foregoing explicit annotation of samples with timestamps and instead associating a sampling period with a WaveSeg, as is also illustrated in the example below. Each WaveSeg contains self-explanatory metadata about the sensor readings such as location, device name, sensor name, sensor id, sampling interval and start time of the readings to enable rich data querying capabilities. Additionally, each channel within a sensor is separately specified with channel name, units for the readings and an array of float values specifying the sensor readings. Additional metadata information, such as location, can also be easily added to this description.

Figure 2.3 illustrates the JSON representation of a WaveSeg, as used by a sensor device to upload data. While WaveSegs significantly improve upon per-sample storage, the number of WaveSegs stored in *SensorAct*'s database nevertheless directly affect query processing performance. To further optimize performance, *SensorAct* opportunistically merges WaveSegs as they are uploaded by a device.

**Guard Rule Engine for Selective Sharing**

*Guard Rule Engine* in *SensorAct* is designed to support *selective sharing* of sensor data and actuation control with users and external applications as governed by the corresponding owner. All access requests for actuators or sensor data are governed by Guard Rule Engine. Tight control on the access is maintained through owner-defined *guard rules* which are policies for restricting access to the data and control of the devices configured for buildings. Guard Rule Engine enables *fine-grained access control* by allowing the owner to define rules based on user, group, time, location, sensor data, and actuators. Every device registered with the VPDS has

```
{
  "DEVICE_NAME": "Office_Flyport",
  "SENSOR_NAME": "MultiSensor",
  "SENSOR_ID": 1,
  "SAMPLING_INTERVAL" : 1,
  "EPOC_TIME": 1344147449,
  "CHANNELS": [
    {
      "NAME": "Temperature",
      "UNIT": "Celsius",
      "READINGS": [28.1,28.2,28.6,28.5,28.2,28.6,28.5,28.7]
    },
    {...}
  ]
}
```

Figure 2.3: Sensor data representation using WaveSeg format.

```
{
  NAME: "ReduceResolution",
  PRIORITY: 1,
  TARGET_OPERATION: "READ",
  TARGET_CONSUMERS: [ "alice@sensoract.edu" ],
  CONDITION: "NOT $(WORK_TIME)",
  ACTION: "DownSample(avg, 15min)"
}
```

(a) Guard rule with macros as condition

```
{
  NAME: "WORK_TIME",
  VALUE: "[ * * 9-18 * * 1-5 ]"
}
```

(b) Macro definition

Figure 2.4: An example of a guard rule and a macro for selective sharing of sensor data.

an associated set of guard rules, created by the corresponding VO, for facilitating external (or shared) access.

Figure 2.4 shows an example of a guard rule and a macro. This guard rule enforces a policy to share data during non-working hours only, which is defined by the macro using a *Cron* time expression and reduces the data access resolution to 15 minutes. The guard rule itself does not necessarily contain references to specific users, sensors, or actuators, but it can be associated with them later by its owner. This *lazy-association* allows users to reuse same rules for different users, multiple devices, or groups of devices.

### 2.2.3 Broker

In *SensorAct*, a trusted broker contains a registry of users, a registry of VPDSes, and it acts as a mediator to assist client applications to establish a connection with multiple VPDSes. A VO needs to register her VPDS on one of the brokers to share data and control with other users registered on the corresponding broker. For data communication to scale across multiple VPDSes and users managed by a single broker, direct communication over a secured channel is provided between users and VPDSes.

### 2.2.4 Applications

Programming abstractions in *SensorAct*, using RESTful APIs, allow easy development of third-party applications that provide a user with controlled access to sensor data and actuators. A *SensorAct* user may have two roles: i) An owner of her own devices and their correspondingly associated VPDS ii) A user with data and control access as per the privileges assigned by the owner of other VPDS. A single user could be the owner of her own VPDS and user for some other VPDS. As an owner, a user may grant controlled access to other users thus letting them access sensor data from her devices or to even control them in a constrained manner. Third-party applications may provide users with a more convenient interface to the underlying functionality of VPDS and broker. We discuss three such third-party applications in Section 2.4.

## 2.3 Scripting Framework

The *scripting framework* in *SensorAct* provides an application execution environment within the middleware for high-level scripting languages in a sandbox. Unlike other systems, wherein external applications read sensor data and perform control actions outside the middleware, this framework enables building owners to inject their custom application logic written in a high-level scripting language, termed as *tasklets*, into the middleware to perform sophisticated energy

management and control operations. These tasklets can be scheduled to read and process live sensor data streams. The proposed *Scripting Framework* also provides a set of read functions to access sensor data streams and write functions to control the actuators. These read/write primitives can then be used to develop complex (one-shot or persistent) control actions providing *rich support for automation* including sensor data processing, data fusion, actuator control, and notifications. Figure 2.5 illustrates the workflow of the proposed scripting framework.

### 2.3.1 Tasklet Workflow

A *tasklet* in the scripting framework is a piece of light-weight non-blocking script that performs a particular set of operations. As shown in Figure 2.5, it consists of *tasklet description* and *tasklet script*. While the *tasklet description* contains meta information about the tasklet, such as resources (sensors and actuators identifier) to be used, parameters, scheduling and triggering conditions, *tasklet script* contains the application logic in a high-level scripting language. More details about tasklet description can be found in [64]. A *tasklet scheduler* receives the tasklet execution requests, submitted using the corresponding tasklet management API provided by the VPDS, and it is responsible for scheduling, managing, and controlling the tasklet throughout its life-cycle. Once a tasklet is scheduled for execution, the *tasklet scheduler* first classifies and schedules the tasklet, based upon the identifiers used in the *When* primitive of the description, as one of the following categories:

***One-shot*:** One-time and immediate execution of a tasklet script. For example, querying the current status of a sensor or instantaneous switching of an actuator.

***Periodic*:** Executing a tasklet script whenever timer elapses to perform periodic operations. For example, switch on my office air-conditioner at 9 AM only on weekdays or email me the electricity usage summary every day at 8 PM.

***Event based*:** Executing a tasklet script whenever a change in the value of sensors or actuators is observed. For example, switching off lights when a window is opened.

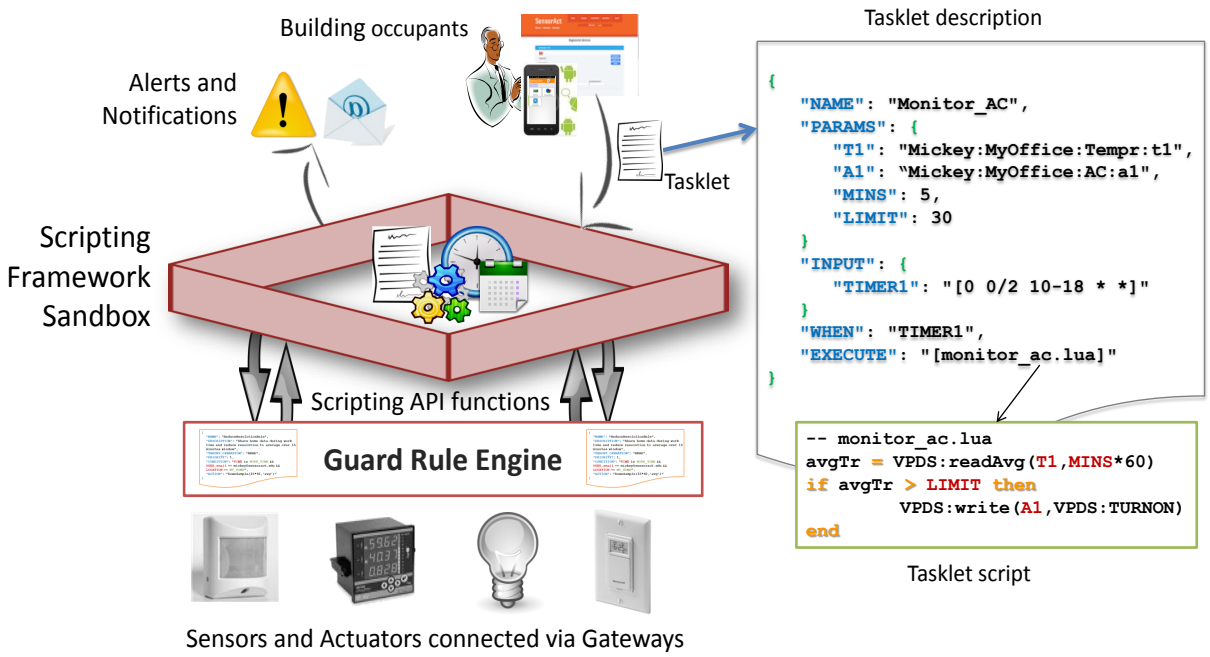



Figure 2.5: Scripting framework workflow with an example Tasklet.

As per our own experiences with implementing several energy management applications and discussion with facility manager, these three tasklet types are sufficient for automating all the energy management tasks in a building.

Tasklet manager provides an isolated execution environment for each tasklet and restricts any interaction among them for sensitive building control applications. By default, a tasklet inherits the privileges of a user who submitted it. All read and write operations on sensors and actuators performed from tasklets go through Guard Rule Engine. Correspondingly, tasklets can only perform operations allowed for a user invoking them, thus protecting against unauthorized access.

### 2.3.2 Tasklet API Functions

The scripting framework in *SensorAct* provides a set of primitive API functions as shown in Table 2.1, in order to perform various runtime operations. Tasklet scripts can invoke these low-level API functions and it enables developers to create and schedule custom building control

Table 2.1: Primitive API functions in Scripting framework available to use in Tasklet Scripts.

```
map  read(DeviceId, duration(in seconds))
map  read(DeviceId, startTime, endTime)
number read(DeviceId, duration, [sum|count|min|max|mean] )
number read(DeviceId, startTime, endTime, [sum|count|min|max|mean] )
string plot(DeviceId, duration)
boolean write(DeviceId, status=ON|OFF|value)
boolean email(to, subject, message [,plot])
boolean sms(to, message)
```

and automation applications. While expert users can write complex tasklet scripts on their own, novice users can use an interactive web interface to create simple automation applications. We also plan to provide *tasklet templates* for commonly used applications, so that users can easily create tasklets by filling in only the required parameters.

The proposed tasklet framework provides a platform to implement and schedule several energy management tasks within the middleware system. Examples are 1) inferring high-level rich occupant-specific context information, such as occupancy and usage patterns, by fusing several raw sensors and actuator values; 2) automation of the routine building control activities performed by the occupants in their day-to-day life, e.g., pre-heating or pre-cooling the workspaces in advance; 3) custom creation of *coordinated appliances* based upon detected building events, e.g., switching off a meeting room may result in switching a group of devices together or in a particular sequence; and 4) development of an automated alert or notification system for monitoring and management of building premises, e.g., alerting the facility management team in case of any abnormal energy consumption. In Section 2.5, we present a list of energy management applications created using the proposed tasklet framework.

## 2.4 Implementation

In this section, we present the implementation details of a prototype *SensorAct* architecture elements, as discussed in Section 2.2.

### 2.4.1 Gateways and Devices

Gateways interconnect existing and new sensors and actuators in the buildings with a VPDS. In the current implementation, three gateways are supported: 1) LabSense [25] for interfacing Z-Wave based ambient sensors (temperature, light intensity, motion, and door contact status) in existing Home Automation Systems, Modbus based smart energy meters, SNMP [54] based Raritan power distribution units [37], and Modbus based Veris [47] and Eaton [9] energy meters, 2) sMAP [82] to communicate with several commercial energy meters and HVAC systems using different protocols such as Modbus and BACnet, and 3) A custom built Wi-Fi based Flyport [17] module to interface ambient sensors and actuation relays. These gateways push sensor readings in WaveSeg format, as described in Section 2.2.2, to VPDS using RESTful APIs, as shown in Table 2.2, and execute the actuation commands received from the building control applications.

In the current implementation, *SensorAct* natively supports only push-based sensors, i.e., sensors pushing data to VPDS using its RESTful interface. However, pull-based sensors can be easily incorporated using third party gateway applications. Currently, we use two such gateway applications - LabSense [25] gateway to pull data from Z-Wave based sensors and a PyModbus based python application to pull data from Modbus enabled smart meters.

### 2.4.2 VPDS and Broker

Various VPDS and Broker components, as explained in Figure 2.1, was implemented in Java using open-source tools and the code is released in open-source for community use [42]. *SensorAct* exposes a rich set of RESTful APIs for most of its functionalities. Such open APIs enable easy integration with other systems and allow developers to write custom third party (stand-alone, web, or mobile based) applications for extending the system features, e.g., visualization, scripting, access control, and sharing. Table 2.2 lists the APIs currently implemented for various components in VPDS and Broker. In the current implementation, we use MongoDB[1] for storing

[1] `http://www.mongodb.org`

Table 2.2: A list of APIs supported by different components of the *SensorAct* architecture.

| **Component** | **VPDS API endpoints** |
|---|---|
| User | `/user/{register\|list}` |
| Device | `/device/{add\|delete\|get\|list}` |
| | `/device/template/{add\|delete\|get\|list}` |
| Guardrule | `/guardrule/{add\|delete\|get\|list}` |
| | `/guardrule/association{add\|delete}` |
| Tasklet | `/tasklet/{add\|delete\|get\|list}` |
| | `/tasklet/{execute\|cancel\|status}` |
| Data | `/data/{upload/wavesegment\|query}` |
| Share | `/device/share` |

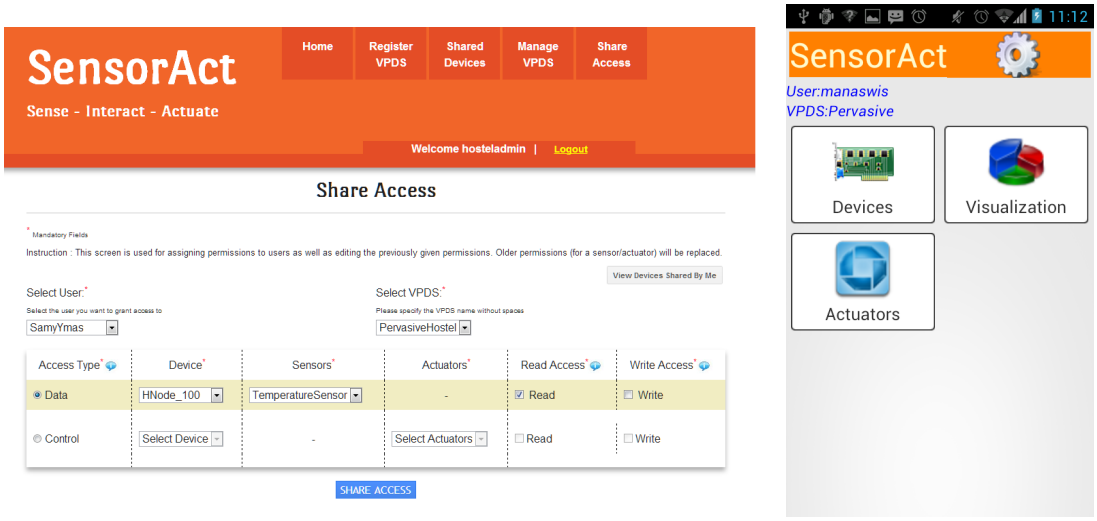
| **Component** | **Broker API endpoints** |
|---|---|
| User | `/user/{register\|login\|list}` |
| VPDS | `/vpds/{register\|get\|list}` |
| Device | `/device/{search\|share}` |
| | `/device/{user\|owner}/shared` |

device profiles, user profiles, guard rules and tasklets due to its flexible, schema-less document format. We support two types of time series sensor data archiver, Informix database[2] on servers for optimized storage and MongoDB on Single Board Computers (SBC) for portability.

The scripting framework in *SensorAct* uses Quartz scheduler [36] library to schedule and execute *tasklets*. It is a high-performance, industrial-strength job scheduler which scales well to manage a large number of tasklets. We leverage its inherent support for cron based periodic jobs to implement periodic tasklets. We also implemented additional modules to support event-based tasklets that require sensory event registration and notification to trigger the tasklets whenever the system receives new data.

The current implementation supports Lua [26] and Jython [24] to write tasklet scripts. Lua was chosen because it is a light-weight, compact, and easy-to-learn scripting language which also has been widely used to program home automation systems. Jython, an implementation of the Python scripting language in Java, provides rich support for data processing. Java to Lua integration was done using jnlua binding engine which is invoked via Java Scripting API framework from the tasklet manager. Several higher level scripting API functions, listed in

[2] `http://www-01.ibm.com/software/data/informix`




(a) Browser application (b) Mobile application

Figure 2.6: Sample browser and mobile application user interface

Table 2.1, are exported into Lua interpreter's execution context so that the API functions can be directly called from the user-written tasklet scripts. Further, a VPDS instance can be hosted on multifarious devices including single board computers such as Raspberry Pi [38].

### 2.4.3 Third party Applications

Three third-party applications were implemented using the VPDS and Broker APIs and they are explained below.

**Web portal:** It is a configurable stand-alone web application, as shown in Figure 2.6a, that interacts with a broker and registered VPDSes using *SensorAct* APIs. Both VPDS owner and other users can use this application to visualize sensor data and to manage devices, guard rules, and tasklets based upon granted privileges.

**Time and presence-based actuation:** It is an intuitive web interface, extended from the web portal application, which allows users to actuate their devices remotely based on time and presence, e.g., pre-heating/cooling a workspace. The user interface makes use of tasklets to allow users to switch their appliance "now", "once" at a specific time, or "periodically" at a regular time

interval. Guard rules are used to restrict users to actuate the devices in their own spaces only. This system is currently under deployment in a commercial building for lighting control.

**Mobile application:** An Android application was developed whereby users can specify their Broker credentials and correspondingly manage the devices owned or shared with them.

Further, in order to simplify the *SensorAct* system installation for a building, all VPDS, broker and user interface components of *SensorAct* were packaged into a Virtual Machine image. The detailed installation instruction manual was documented and the usage of the system was evaluated using a user study (See Section 2.5.3).

## 2.5 Evaluation

The proposed *SensorAct* architecture is evaluated based on multiple real-world deployments for supporting different usage scenarios. Particularly, the utility of the proposed tasklet framework is shown for various energy monitoring and alerting applications. Further, a user study was performed to evaluate the different aspects of deployment of *SensorAct* in a student dormitory building.

*SensorAct* was deployed in four different settings as illustrated in Figure 2.7: (1) Campus-wide energy monitoring at IIIT-Delhi, India, (2) Student dormitory deployment at IIIT-Delhi, India, (3) Research lab at UCLA, USA, and (4) Research wing at IIIT-Delhi. Each deployment was done with different requirements, devices, gateways, users, and scales as shown in Table 2.3. Separate VPDS instances were used for each deployment, and they were registered with a common broker, hosted at IIIT Delhi, for sharing sensor data and control with users across different buildings. In each deployment, the corresponding owner or tenant of the building managed the VPDS and granted privileges to other occupants who all were registered themselves with the common broker if required.

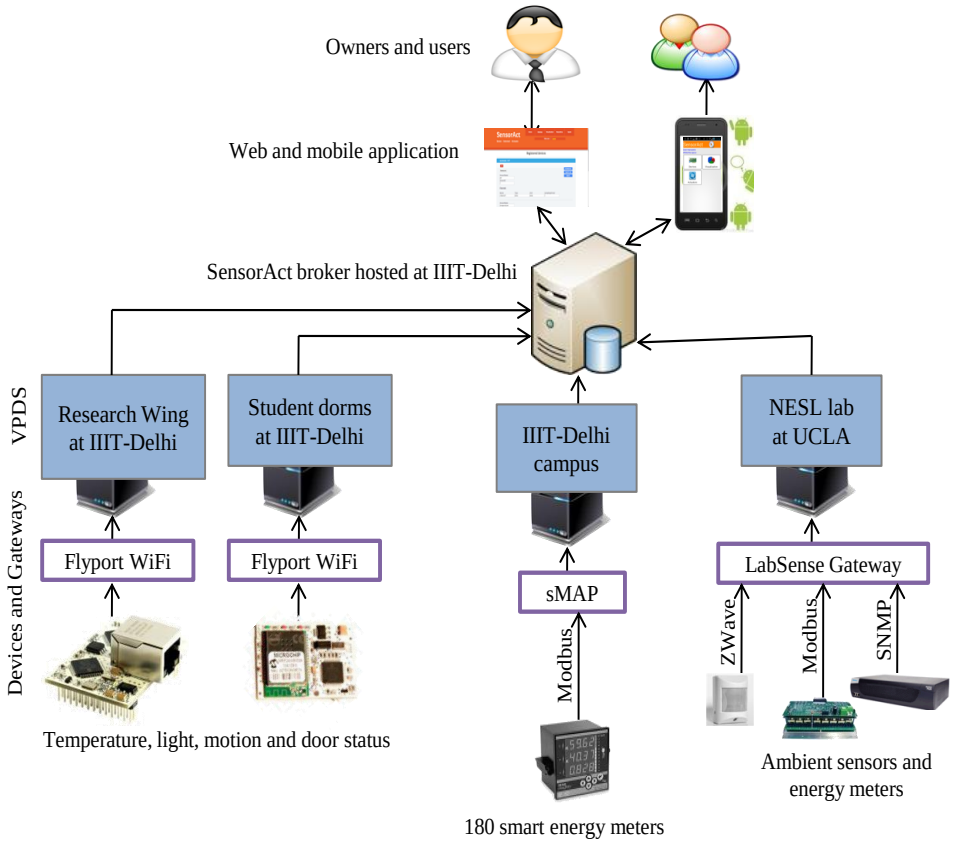



Figure 2.7: Deployment scenario of *SensorAct* architecture showing VPDSes deployed across four different locations and a common broker hosted at IIIT-Delhi.

### 2.5.1 Campus Wide Electricity Monitoring at IIIT-Delhi

One of the largest deployments of *SensorAct* involved monitoring the electricity usage of all the buildings in IIIT-Delhi campus. 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 flats), 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 sMAP based archiver was used for collecting meter readings at every 30 seconds. Existing BMS subsystems, such as HVAC, were also interfaced with the sMAP archiver using BACnet

Table 2.3: Different deployment details of *SensorAct* system. Ambient sensors include Temperature, light intensity, motion, and door contact status

| **Deployment** | **Research Wing** | **Student Dorms** | **IIITD Campus** | **NESL, UCLA** |
|---|---|---|---|---|
| Purpose | Occupancy sensing and data sharing | Occupancy sensing and data sharing | Energy monitoring and alerts | Occupancy sensing and energy monitoring |
| Scale | Single building | 21 student dorms | 6 buildings | Research lab |
| Platform | Virtual machine | Laptop and PCs | Virtual machine | Virtual machine and MiniITX |
| Sensors | Ambient sensors (14) | Ambient sensors (21) | Electricity meters (180) | Ambient sensors and electricity meters (3) |
| Sampling rate | 1 second | 1 second | 30 seconds | 2 seconds |
| Gateway & protocols | FlyPort, WiFi | FlyPort, WiFi | sMAP, Modbus | LabSense, ZWave, Modbus |
| Duration | 2 months | 1 month | 4 months | 8 months |
| Users | 20 | 21 | 2 | 15 |

and Modbus bridge. A separate sMAP to *SensorAct* gateway was implemented for uploading all real-time measurements to *SensorAct*. As per FM's requirements, three energy management applications were created, to be managed by him, using the proposed tasklet framework.

### Abnormal street light usage detection

IIIT-Delhi campus contains pathways around the campus for about 2 kilometers. There are about 135 street lights installed in the path way. These street lights are manually switched on in the evening and switched off in the morning by an operator. They consume over 6 kilo-watts of power. From the street light meter readings, as shown in Figure 2.8, we observed that there were some suspicious electricity usage events during day time and occasional events during night time as well. Hence, to monitor such abnormal electricity usage events by street lights, two tasklets, one for day time and another for night time were setup. They computed average electricity consumption of the street lights every five minutes. If the average consumption was above a threshold (derived based on our observation), these tasklets sent an email and SMS to the FM for taking necessary action. Over the course of past one month of this setup, these tasklets

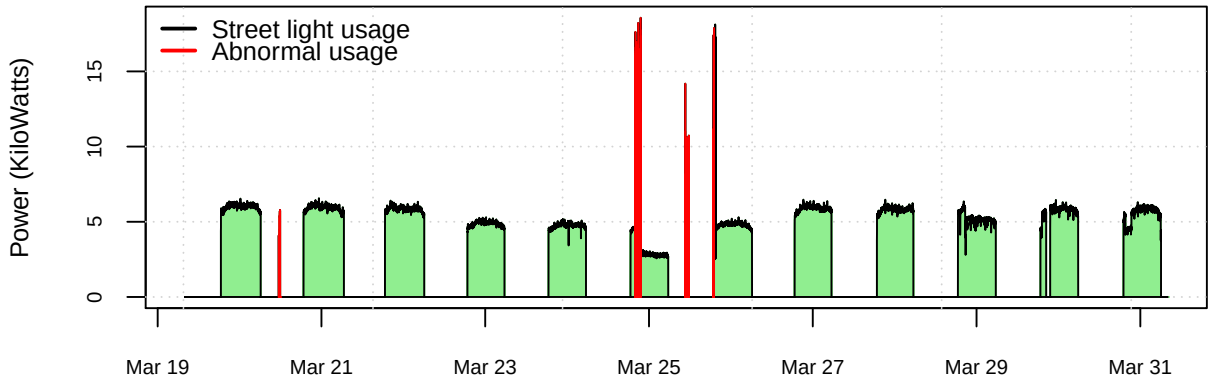



Figure 2.8: Daily electricity usage pattern of street lights (6:30pm to 6:00am every day) for 12 days. Some abnormal energy usage events are marked in red.

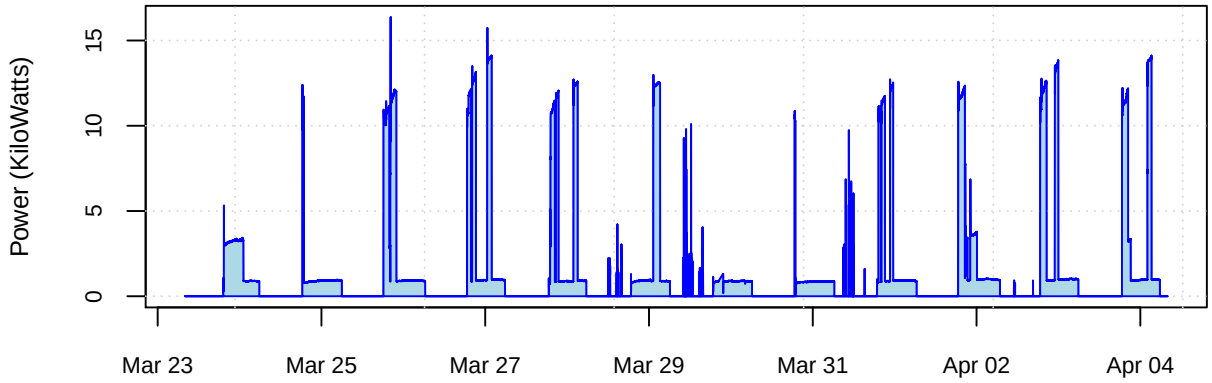



Figure 2.9: Electricity usage patterns of sports area lights for 12 days. Few street lights around the sports area consuming about 0.9 kilo-watts of power are used from 6pm to 6am every day. Each spike in this plot corresponds to the usage of flood lights in the sports area.

detected two such abnormal electricity usage events and notified the FM. The FM asked the facility support team to check the street lights and its power meters. FM also suggested that such period monitoring tasklets for street lights are essential, particularly in India, as electricity theft is not uncommon. Hence, the implemented tasklet helped the FM for continuous monitoring of street lights and it prevented any electricity theft.

### Sports area usage summary

IIIT-Delhi campus has a sports area that consists of a basketball court, a football court, and a common playground. The entire sports area is equipped with several flood lights and they

consume over 10 kilowatts of power when they are in use. At present, students are advised to turn on and off these lights whenever they want to play during night time. For accountability and to make policies for the sports area usage, FM wanted to know how many hours these sports area was being used every day and the corresponding energy usage.

Since the sports area lights were connected through a separate smart meter, we monitored the electricity usage for a month to know the baseline usage. Figure 2.9 illustrates power consumption of the sports are lights (all the peaks) along with some other constant load for 12 days. Based on our observation, a periodic tasklet was created and it was scheduled to run at 8 A.M. every day. The tasklet reads the smart meter's readings for the previous night and filtered out the readings only for the sports area usage based on a known threshold value. The tasklet is also configured to send a summary email to the FM about how many hours the sports area was used and the total energy consumption.

**Critical energy usage alert**

IIIT-Delhi campus receives two power lines from the grid, one for commercial and another for residential usage. While the residential power line is being used for faculty apartments and student's hostels, a commercial power line is being used by the rest of loads in the campus. External commercial load from the grid is stepped down using two transformers, one for supplying high-voltage loads such as HVAC, and another one for connecting commercial usages such as lighting and IT devices. As shown in the Figure 2.10, commercial energy usage for a typical working day is over 3,000 units and it varies based on many other factors. In order to monitor the overall energy usage, FM wanted to develop an application that can alert him when the total energy consumption of the current day exceeds the previous day.

A periodic tasklet was created for monitoring the campus-wide energy usage in real time. The tasklet was scheduled to be run at every five minutes. The tasklet script was configured to perform the following tasks in each run: 1) it reads the previous day and current day energy usage from the

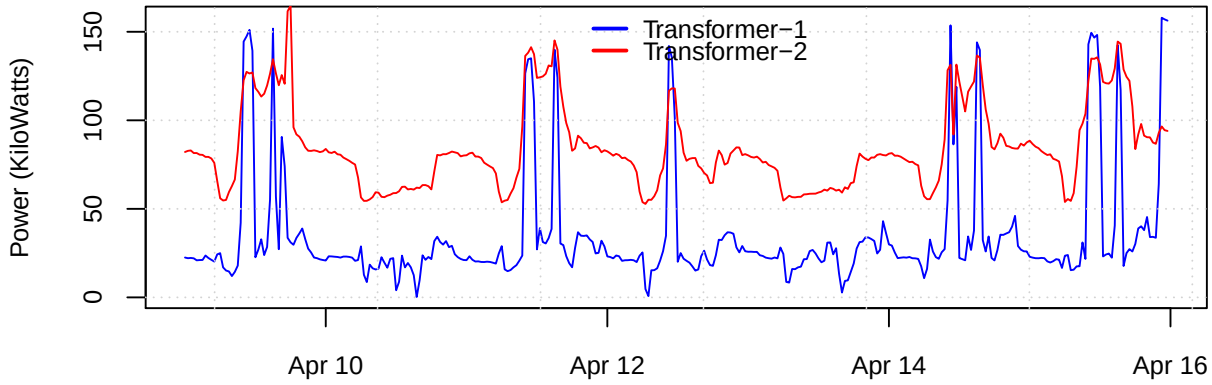



Figure 2.10: Campus-wide total commercial electricity usage (from two transformers) for 1 week. The spikes in transformer 1 corresponding to the electricity consumption of HVAC systems, that consumes over 100 kilo-watts.

smart meters which are connected with the corresponding transformers, and 2) it compares them and sends an email and SMS to the FM if the current day energy usage exceeds the previous day consumption, 3) Since this is a periodic tasklet and to avoid continuously sending the notification on successive alerts, it was configured to send alerts two times maximum in a day.

### 2.5.2 Research Lab at UCLA

In this deployment, *SensorAct* was deployed for a 1,200 square feet university research lab, occupied by 12 graduate students, in UCLA for occupancy and electricity monitoring applications. It was instrumented with several sensors including 1) high-frequency multi-channel Raritan, Eaton, and Veris power meters measuring several electrical parameters of the lab servers, devices, and power outlets, and 2) Z-Wave based Aeon Labs door sensors and HomeSeer [19] multi-sensor (measuring motion, temperature, and light intensity level).

LabSense [25] gateway was used to pull data from all of these sensors and push it to *SensorAct*. FireSense [16] system was also used to monitor the real-time network traffic of the lab and the network event logs were pushed to *SensorAct* in real time. A separate VPDS instance was hosted on a VM and it is collecting data for the past 6 months. The VPDS was registered with a common broker hosted in IIIT-Delhi. Permission to access sensor data and create tasklets

was shared with one of the graduate student at IIIT-Delhi, who created a computed sensor for presence detection (discussed in Section 2.4) and performed occupancy based experiments.

*Occupancy and energy usage summary:* Based on this setup, a periodic tasklet was created for monitoring the occupancy and electricity usage of the lab. The tasklet was scheduled to run at 12 A.M. midnight every day to send a summary report to the lab members. The tasklet script was primarily doing three functions: 1) it decided whether some one is entering or exiting the lab by fusing the door status (open/close) with motion sensor events, 2) it compares the entry and exit events of the two doors and decides the first entry and last exit time of the lab, 3) it aggregated the total power consumption of a day and created a plot of power consumption at 5 minutes, and 4) finally, it sent a summary email mentioning how many hours lab was opened on a particular day and how was the total power consumption during that time. This energy usage summary report was useful for giving insights about the usage patterns of the lab and for increasing energy usage awareness of the occupants. Further, the inferred occupancy information can also be used to pre-heat/cool the lab.

### 2.5.3 Student Dormitory Deployment at IIIT-Delhi

In this deployment, in order to validate the utility and easy deployment of *SensorAct* middleware system, 21 student groups (each with 2 students at IIIT-Delhi were engaged. They deployed Flyport based Wi-Fi nodes in student dormitory rooms, collecting motion, temperature, and window status information every second, by following the installation manual. One of the faculty coordinators hosted a central VPDS on one of the servers and registered it with the broker at IIIT-Delhi. Students were then asked to register themselves and room occupants (for the case when they are deploying in someone else's room) on the broker. Faculty coordinator first shared privileges with the engaged student groups and allowed them to create devices in his VPDS though they were not allowed to see data from the devices added by them. This privilege was revoked after two days, and the added devices were shared with the corresponding room occupants to let them

decide whom they want to share data. While the study mandated each student to collect only 2 days worth of data from their individual deployment, a total of 100 days worth of data together from all the groups was reported by the students.

A survey was conducted at the end of the deployment to get feedback from these students on different aspects of using the *SensorAct* system. Overall 17 student groups responded to the survey at the end of the study. Among them, 16 student groups had no prior experience with uploading data to a server or cloud system such as *SensorAct*. More than 80% of the respondents mentioned that *SensorAct* installation and configuration on their individual laptops, with different OS, was easy. Approximately 45% of respondents gave their preference for local hosting of *SensorAct* VPDS instead of the cloud, due to privacy reasons, if they were to deploy *SensorAct* for monitoring and control in their homes. Students used *SensorAct* for basic data collection and visualization while the occupants used sharing capability for data and control to provide access of their devices to their friends. 64% of them found *SensorAct* to be a good and usable system. More than 90% of responses rated *SensorAct* documentation to be detailed enough with 73% of them asserting that, with the current level of documentation, a new person can setup *SensorAct* system without any help. More than 90% of the responses were positive about the usability of the browser application that was used as a front-end for the study.

## 2.6 Related Work

We group the related work into following three categories: building management systems, research prototypes and architectures, and cloud services.

### 2.6.1 Building Management Systems

Existing BMS [20, 23, 43, 45] and home automation systems comprise of several isolated subsystems each performing a particular task such as fire alarms, security and access control and

HVAC. While they include a large number of sensing points spread across a building, data from these sensors are usually inaccessible to building occupants as these systems are normally controlled and managed by a central facility department. While many commercial buildings already have some form of BMS in existence, *SensorAct* can be used to augment them to develop novel occupant-centric applications such as personalized control of workspaces. Gateway applications can be easily developed to interface such M2M applications with *SensorAct* (as shown in Figure 2.1). Further, higher costs typically associated with such BMSs, prohibit their usage across small deployments such as in residential homes. Home automation systems, try to fill in the gap providing comfort to the occupants, using local storage and several automation scripts. However, they provide limited support for fine-grained data and control sharing across multiple homes and users, supported extensively in *SensorAct*.

### 2.6.2 Research Systems and Architectures

Several research systems pertaining to connecting and sharing sensors at Internet scale, such as SenseWeb [90], SensorWeb [80], GSN [59], and WattDepot [72] have been developed and deployed in the recent past. However, these systems are limited primarily to sensory data aggregation and visualization and provide minimal sharing capabilities. Some research software systems have been proposed in the literature that provides an abstraction over diverse devices and enable uniform interfaces to access them [84, 95]. For example, HomeOS [84] addresses the interoperability and usability issues by providing a PC-like abstraction over the networked devices as peripherals for both users and developers. Sensor Andrew [111] shares some common design goals with *SensorAct*. It focuses on a large-scale data aggregation from diverse sensors and event-based control for building operations. However, the data and control sharing mechanism is at a coarse-grained level, based only on user identity. BuildingDepot [63] focuses on managing network of buildings by isolating the data, users, and privilege management from each other.

Similarly, Building Operating System Services (BOSS) and Building Application Stack [83] enable writing portable and fault-tolerant applications on top of diverse physical resources present in buildings. Specific to residential buildings, VHome [115] provides an isolated application execution environment for data analysis. Though VHome shares several design goals with *SensorAct*, it provides an application runtime to execute Cloud Based Application, focusing only on energy data analytics. Further, access control mechanism in VHome supports only time and location-based energy data sharing, whereas *SensorAct* supports fine-grained sharing.

### 2.6.3 Cloud-based IoT platforms

Several public cloud-centric IoT platforms exist today for collecting, archiving and visualizing the real-time sensory data such as Xively [55], Nimbits [52], Mango M2M [51], Sen.se [53], SmartThings [58], openHAB [57] and Kaa [56]. While these services provide rich support for data aggregation and visualization of sensor data collected from diverse devices, they provide inadequate or limited support for IoT applications specifically pertaining to the buildings domain: 1) They provide limited capabilities for sharing the collected sensor data as they follow "all-or-nothing" model based only on user identity. Since the sensor data collected from buildings carry several forms of occupancy and usage patterns about the buildings, devices, and occupants [79], controlled and fine-grained sharing is required to protect the sensitive sensory data collected from buildings. 2) They provide limited or no support for controlling the devices and automating their operations. For example, Sen.Se provide remote actuation support but they handle simple use cases wherein the control is manual or is based on events, e.g., whenever motion is detected, switch on/off an appliance.

SmartThings [58] provide scheduling APIs which are limited to creating only one-time and repeating schedules, but not event-based which is necessary for many energy management tasks in buildings. Further, SmartThings's SmartApps model is inflexible for passing custom parameters (see PARAMS clause in the Tasklet) and sensor data events to the application logic. open-

HAB's Rules feature shares the some of the design principles of Tasklets by providing trigger-action based automation. But, it combines the rule configuration and scripts in a single file, making it inflexible for customizing the application logic. In contrast, *SensorAct*'s Scripting framework decouples the tasklet configuration and logic, thus provides flexible, expandable and customizable ways for automating the devices.

The *SensorAct* architecture allows for easy support of complex energy management applications using the scripting framework. In addition to the features provided by these systems, *SensorAct* supports rule-based selective sharing model for sensor data. Further, the proposed scripting framework (discussed in detail Section 2.3) supports notification and enables users to create *Computed Sensors* (e.g., occupancy) that can then be further used to create automation scripts depending on the context and not on real sensor values. *SensorAct* scripting framework is also flexible to be extended for supporting any new scripting language which is not the case with any of the existing systems.

## 2.7 Summary

In this chapter, we described the design, development, deployment, and validation of *SensorAct* system for optimizing the energy usage in buildings. *SensorAct* system is a result of the needs informed by our first-hand experience in monitoring building ecosystem: an open, flexible, extensible, and scalable information substrate for buildings into which multitude of sensors, actuators, and applications could be integrated.

*SensorAct* architecture supports several novel features including (i) Virtual Personal Device Servers (VPDS) for local hosting the middleware within the building, (ii) Scripting framework within the system for providing rich support for developing and automating energy management applications, and (iii) A rule-based access control mechanism for fine-grained sharing of sensor data and actuation control with other users. Validation of the developed system was done using multiple deployments, from residential to commercial buildings, spread across India and USA.

The *SensorAct* system enabled the development and deployment of extensible building management applications. Using our prototype implementation of the system, we experimented a number of real-world energy monitoring applications for identifying the potential energy wastage events in our testbed buildings, by applying simple threshold-based rules. Threshold-based rules, which operate directly on sensor data in real-time are simple, but they are inadequate in detecting complex aberrations under different building context, e.g., a sudden increase in the energy usage of a room is due to genuine reasons (high occupancy) or device misbehavior? To address this, in the next chapter, we present a system for identifying the operational context of buildings using sensor data analytical methods, which enables the identification of context-based aberrations.

# Chapter 3

# OpenBAN – A Context Inference System for Smart Buildings

In the previous chapter, we described the architect of *SensorAct* system for developing extensible building applications and showed that how the simple threshold-based rules can be applied for detecting energy usage aberrations. Though the rule-based method is simple and powerful, it is insufficient in detecting complex context-based aberrations, which require advanced sensor data analytics for identifying the operational context of the buildings e.g. operating state of a device, and occupancy levels of a building region. In this chapter, we present the design and development of *OpenBAN*, a middleware system service for sensor data analytics to detect complex context-based aberrations in buildings.

*OpenBAN* 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* facilitates the inclusion of several analytical algorithms as part of the *sense-analyze-act* pipeline for developing novel building energy management applications. Using our prototype implementation,

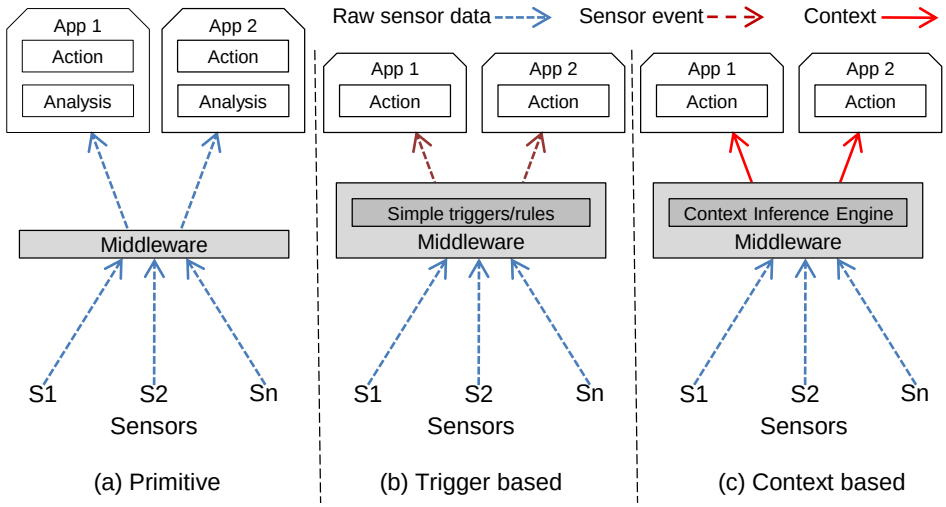



Figure 3.1: Evolution of building middleware systems based on support for processing sensor data. (a) primitive middleware provides no support for sensor data analytics, (b) rule-based middleware provides trigger-actions based on thresholds, and (c) context-based middleware provides sophisticated analytics for inferring context from sensory data.

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.

## 3.1 Background and Motivation

### 3.1.1 Evolution of building middleware systems

Diverse sensor data streams generated in building management subsystems are often discarded after they are processed by 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 [84], *SensorAct* , BuildingDepot [119], and

BOSS [83]. 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 [84, 98, 119].

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 [98], occupancy prediction [101], and energy monitoring [115]. 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 3.1.

*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[84], *SensorAct* , and BuildingDepot [119] are examples of primitive support category.

*Trigger-based* middleware systems, on the other hand, allow 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

Table 3.1: Motivating energy management applications which require complex features and analytics on top of the collected sensor data.

| **Application** | **Example features** | **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 | SVM |
| Indirect occupancy sensing | Mean, min, max, and standard deviation of network activity | SVM |

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 [46] 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 [75] and occupancy detection [86, 101]. 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 3.1. 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.

### 3.1.2 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.

**1. Energy disaggregation:** Previous studies have shown that providing appliance level consumption feedback to consumers can help them save upto 15 % energy [81]. 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) [93] 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.

**2. 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."*[107]. While the law exists, the compliance is not good and enforcement non-existent. 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.

**3. Indirect occupancy sensing:** Occupancy sensing is an important research problem in building energy domain due to its role in several building control applications including HVAC and lighting control [61, 85, 86, 101, 113]. While occupancy information can be inferred directly using several direct sensors such as motion, many indirect sensing techniques have been proposed in the literature using readily available data (collected by service providers such as utilities) such as smart energy meter readings [76, 97] and network activity [112, 117]. In this application, we identify the number of occupants of a building region using the network traffic traces data which require sensor data analytical methods.

Table 3.1 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.

### 3.1.3 Deployment scenarios

Having discussed the motivating applications and the requirements imposed by them, we now look at scenarios in which *OpenBAN* needs to be deployed in. We envision that the proposed *OpenBAN* can be deployed under two different scenarios as follows:

**As an embeddable service with home automation system:** Existing home automation applications such as Vera and HomeOS are often limited to detecting only simple contexts. e.g., "if garage door is left open notify me". In such residential settings, *OpenBAN* can be hosted along with an existing HAS controller software for detecting complex contexts from existing ambient sensors and energy meter data. Similar to developing home automation system *plug-ins*, an analyst or a developer (who has knowledge about analytical algorithms) can create *contextlets* and share them with other home owners. *OpenBAN* can leverage the existing HAS system and build richer context inferences on top. For instance, an example contextlet- "notify when air-conditioner is used when no one is in home" can combine the relatively simple garage door context supported by the HAS with the more complicated energy disaggregation systems (discussed in Section 3.4) supported in *OpenBAN*.

**As a private service for commercial buildings:** For commercial building settings, *OpenBAN* can be hosted along with modern building management systems such as *SensorAct* and BOSS[83] for providing analytics capabilities to novel building control applications. *OpenBAN* can integrate existing sensor data streams and apply analytical functions to extract inter-

Table 3.2: 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. |

esting contexts. Facility managers can create *contextlets* with the help of building data analysts and configure them for multiple buildings. Example applications are energy forecasting, anomaly detection, occupancy detection for controlling lighting and HVAC systems.

## 3.2 OpenBAN System Architecture

Based on the specification of our motivating applications discussed in Section 3.1.2, Table 3.2 lists the functional requirements of *OpenBAN* and its corresponding architectural components that meet the identified requirements. As shown in Figure 3.2, *OpenBAN* middleware consists of four major components: (i) *Data adapters*, (ii) *Feature repository*, (iii) *Model repository*, and (iv) *Context Inference Engine*.

### 3.2.1 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 *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* [18] and Xively [55], 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.

### 3.2.2 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.

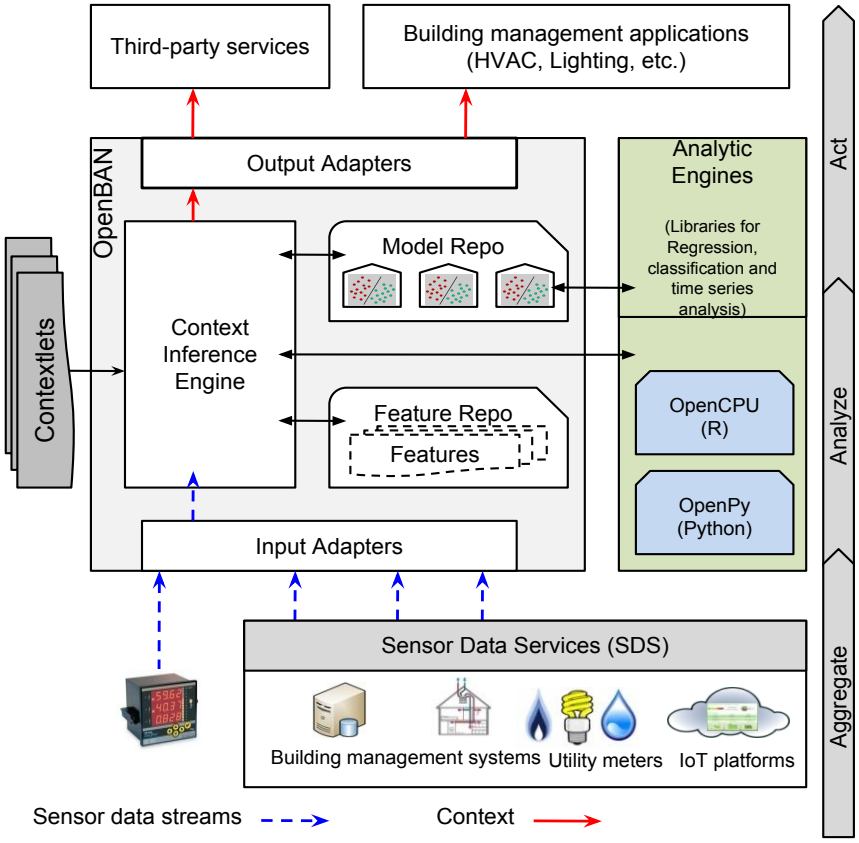



Figure 3.2: *OpenBAN* system architecture showing various system components.

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 3.3 lists a 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 [66, 75, 101]. Additional features can also be computed from these basic features e.g. ratios of different quantities.

Table 3.3: 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* 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` | Minimum light intensity level of a room | | |
| `max` | Maximum temperature of a workspace | | |
| `mean` | Hourly average electricity usage | | |
| `median` | Median $CO_2$ level in an hour window | | |
| `sum` | Total water consumption of a day | | |
| `count` | Number of times door closed/opened | | |
| `range` | Temperate range of a workspace in a day | | |
| `mode` | Mode of a list of active network ports | | |
| `stddev` | Standard deviation of gas usage | | |
| `var` | 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? | | |

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*}}[97]. Similar to feature names, feature templates for a specific application can also be created and shared with other users.

### 3.2.3 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 commonly used for inferring various context information, as listed in Table 3.3, 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 an HTTP API endpoint in an Analytics Engine (see Section 3.2.4) 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 3.3.4). Additional analytics functions, developed by *OpenBAN* users, can be made accessible to other users by updating the repository with the required meta information.

### 3.2.4 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 3.2, 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 the 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 3.2.3) for later use.

### 3.2.5 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 3.3 illustrates the workflow of both the training mode and the execution mode.

#### 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*,

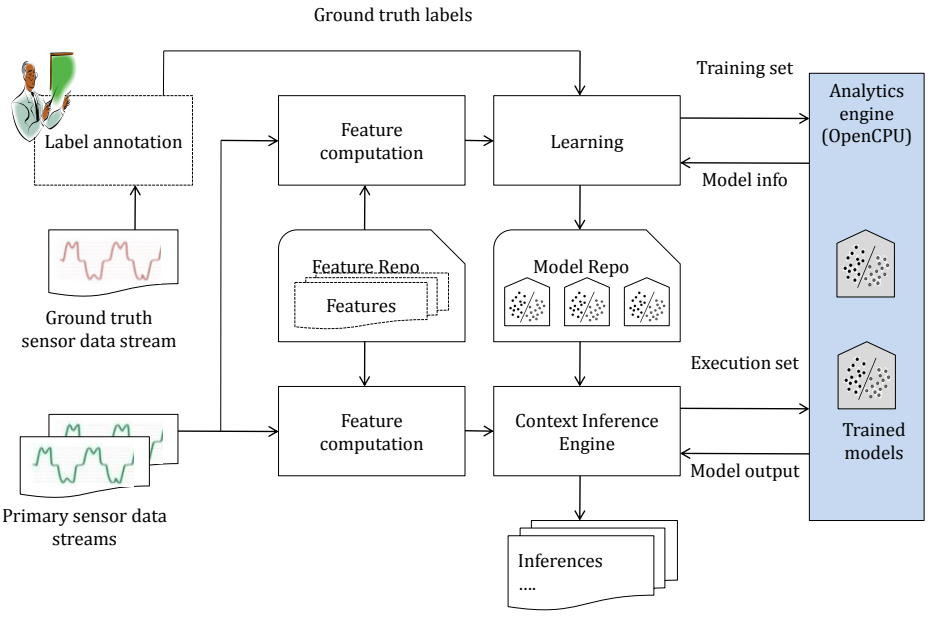



Figure 3.3: The workflow of the Context Inference Engine for training and execution mode.

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*.

### 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.

## 3.3 Implementation

In this section, we describe the details of a prototype implementation of *OpenBAN* architecture, as described in Section 3.2. We have leveraged several open source technologies for our implementation, and have released the code as open source [29]. We have used Java-based Play framework [35] 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 3.2.5 uses Quartz scheduler [36] to schedule and execute the previously learned models.

### 3.3.1 Data Adapters

We implemented three data adapters in the released version of *OpenBAN*: (1) Xively IoT platform [55], (2) sMap[82], and (3) *SensorAct* , 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*.

### 3.3.2 Feature and Model Repository

The released version of *OpenBAN* contains all the features listed in Table 3.3. 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 is 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.

### 3.3.3 Analytics Engine

*OpenBAN* leverages OpenCPU [30] 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 a *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 algorithms is also included in the model repository. In addition to OpenCPU, we have implemented our own analytics engine called OpenPy [33] 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.

### 3.3.4 User Interface

We implemented a web interface for *OpenBAN*, as illustrated in Figure 3.4, 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. The 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.

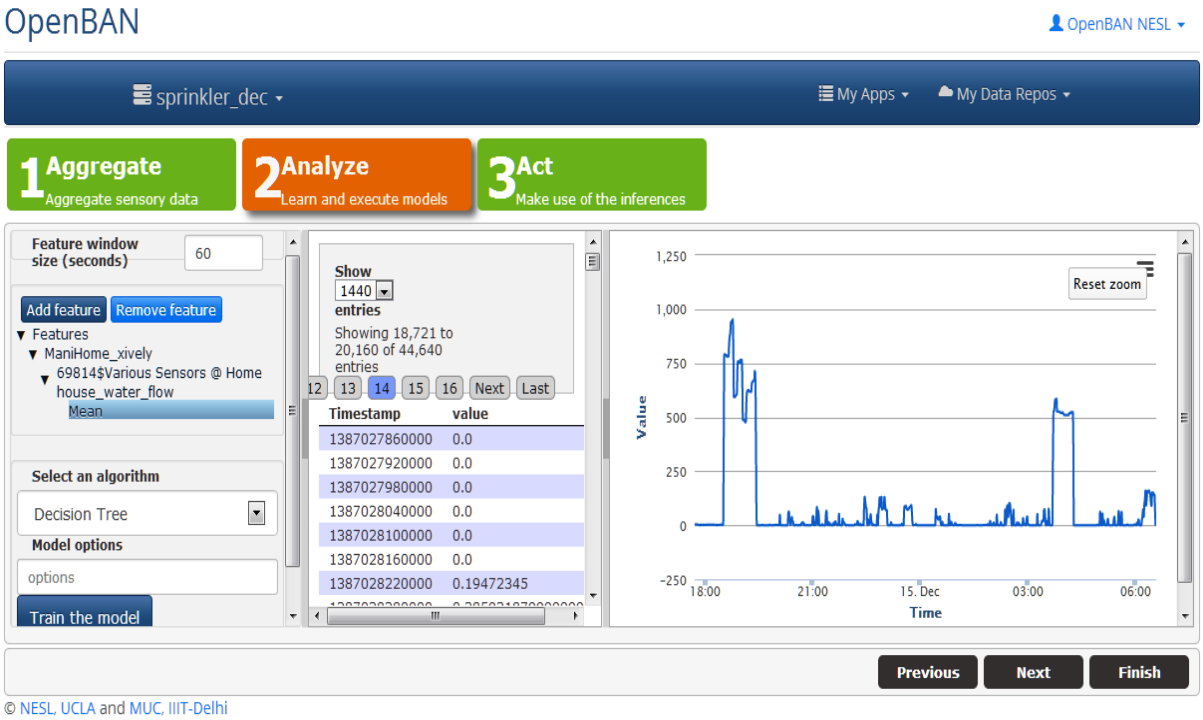



Figure 3.4: *OpenBAN* user interface showing the "Aggregate-Analyze-Act" pipeline for a sensor data analytical application.

**Data adapter instantiation**: In order to integrate a sensor data service into *OpenBAN*, the user first creates an instance of a particular *Data adapter*. Such instantiated data adapter is called a sensor *Data Repo*sitory. The 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 is referred back whenever *OpenBAN* needs to read or write the sensor data to 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 possibly be stored in 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 12 am. Based on the user-specified information, *OpenBAN* will initiate the execution mode in the background, as discussed in Section 3.2.5.

**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.

## 3.4 Experimental 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.

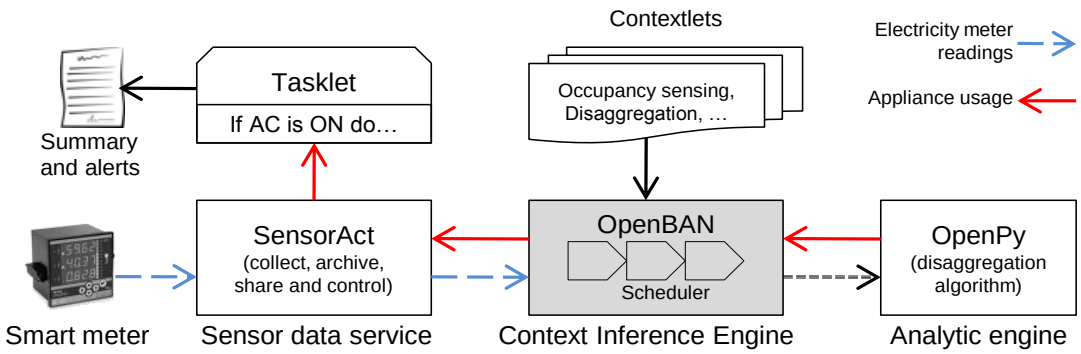



Figure 3.5: Integration of *SensorAct* and *OpenBAN* systems for energy disaggregation application.

### 3.4.1 Energy disaggregation

We leveraged the combinatorial optimization based NILM algorithm, integrated with NILMTK [65], to detect the 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 [65]. Figure 3.5 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[82] 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 set up 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.

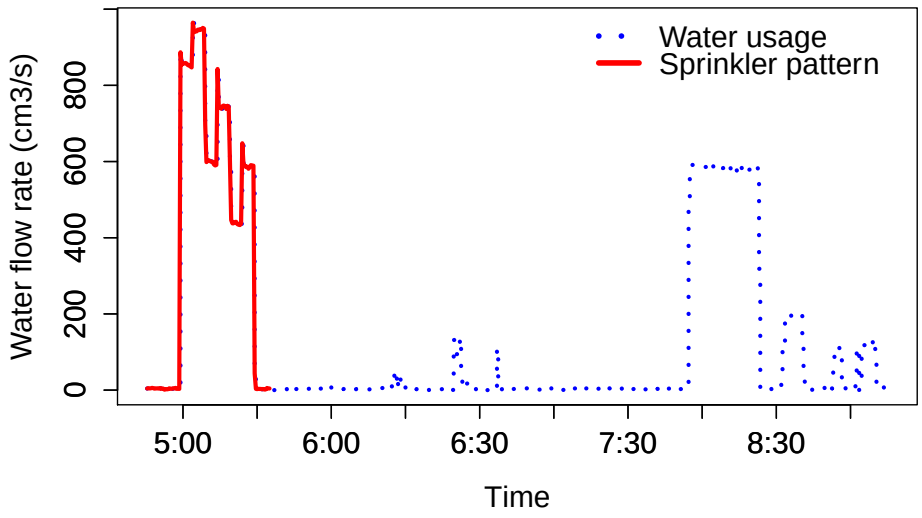



Figure 3.6: Water usage pattern for six sprinkler stations

### 3.4.2 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. Homeowners 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 3.6 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 [28], 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 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.

### 3.4.3 Indirect occupancy sensing

In this experiment, we show the usage of *OpenBAN* to infer the number of occupants in an academic research lab from the live network traffic data. The lab is equipped with a firewall which runs pfSense [34]. A script that invokes the Tshark network analysis tool monitored the network traffic data such as, (i) number of connected peers; (ii) total number of TCP-out, and TCP-in packets, and iii) activities on different standard ports. The script pushed the event logs into *SensorAct* .

We collected the ground truth information using two IP cameras that take images whenever the lab door is opened or closed. Manual observation from collected images was used to count the total number of occupants at every 1-minute interval for 10 days. The first 5 days were used for training. We created a *contextlet* for occupancy sensing in *OpenBAN* and selected the network

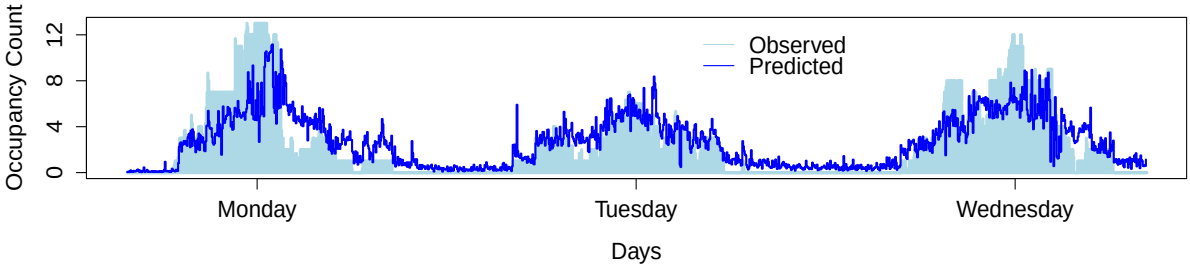



Figure 3.7: Indirect occupancy count prediction from network activity using SVM classifier.

sensor data streams from *SensorAct* data adapter. We selected the features `mean, min, max` and `stddev` from the Feature Repository and SVM classifier from Algorithm Repository. We then used the remaining three days data (excluding 2 days of a weekend) to test the trained model.

The inferred occupancy count data was stored into Dropbox for further analysis. Figure 3.7 illustrates the difference between the predicted and the observed occupancy count. The prediction accuracy of this approach was 84%. This *contextlet* can be scheduled at regular interval for real-time lab occupancy information. This case study shows how *OpenBAN* can be useful for developers and researchers to experiment and create new context identification methods.

## 3.5 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 [31]. For experimental purpose, these

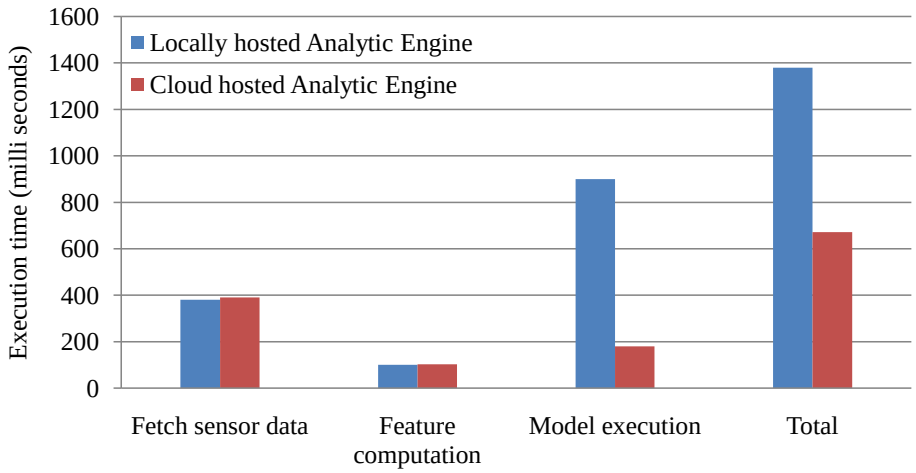



Figure 3.8: Comparison of execution time between local and cloud hosted analytics engine for the energy forecasting *contextlet*

*contextlets* were scheduled to be run at every minute interval. In their each instance of execution, they read past 1-hour smart meter readings and invoke the forecasting model which was learned 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 3.8 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 show that upper time limit for each *contextlet* instance which is about 700 milliseconds for cloud and 1350 milliseconds for locally hosted analytic engines.

## 3.6 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 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 [119] provides a dedicated *Application-Service* 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* system, as described in Chapter 2, 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 a multitude of sensory data and allows applications to subscribe to the desired contextual event and perform actions thereof.

HomeOS [84] presents existing networked devices in homes as PC peripherals. Applications can read the device status or subscribe to events of interest. In either way, the underlying layers pass raw device data to the applications. On the other hand, BOSS [83] 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 transforming 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 [45], 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 [98]. 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 the garage door is left open then notify the user". Mango Automation [51]

**IoT Platforms:** A number of IoT platforms have emerged recently for interconnecting sensors to the Internet and developing novel applications. Examples include Xively [55], Sen.se [53], 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 [49]. 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 [22] provides support for creating home automation recipes using WeMo [48] 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 [99] and Darwin phones [102]). *OpenBAN* shares similar goals with Auditeur [106] 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 [66, 101] 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 [92] and cloud-based general purpose analytics platforms such as BigML [4] and OpenML [32] 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 a real-time continual operation on time series sensor data. Further, all these services require features extracted from raw sensory data as their input. Esper [15] 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.

## 3.7 Summary

In this chapter, we presented *OpenBAN*, an open sensor data analytics middleware which facilitates the development of context-based aberration detection 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 performed a preliminary system performance evaluation and find it to be satisfactory for the intended application.

# Chapter 4

# A Scalable Aberration Detection Technique for Smart Energy Meters

In the previous two chapters, we explained software system based solutions that enable the development and deployment of extensible energy management applications. In this chapter, we shift the focus to proposing novel data analytics method for identifying abnormal energy usage for a network of buildings within a neighborhood. The proposed method leverages the opportunistically available aggregate level smart meter readings and metadata attached to the meter identity, as collected by utilities. We proposed an unsupervised aberration detection method using partial context information which reduces false positives. Further, it gives a ranked list of potential abnormal energy usage instances for further inspection by the building owners [1].

## 4.1 Background and Motivation

There has been rapid growth in energy monitoring infrastructure in several countries. For example, by 2016, 70.8 million Advanced (smart) Metering Infrastructure (AMI) installations are done done by the US electric utilities [110]. Aggregated level electricity usage of both residential and

[1]Please note that, in this chapter, we use the terms aberration, abnormal and anomaly interchangeably, consistent with the literature[67, 88, 114], although their exact definition may differ under different context.

commercial buildings are measured and recorded by the smart meters at a minimum of hourly intervals. Analyzing this massive volume of smart meter data for identifying energy wastage events is an important enabler for reducing the energy usage for a large number of buildings.

Several automated methods have been proposed in the recent literature for identifying abnormal energy usage events from the aggregated level smart energy meter data [67, 88, 114]. One of the important drawbacks of existing anomaly detection algorithms is that various unknown context variables, such as seasonal variations, can affect the energy consumption of buildings in ways that appear anomalous to existing time series based anomaly detection algorithms. Moreover, the primary intuition behind these approaches is that deviation from baseline energy usage bears a one to one relationship with anomalous events. However, this assumption has two major flaws. First, not all deviations from baseline patterns are anomalous; energy consumption can vary significantly based on the day of the week, the time of the year, and many other seasonal and contextual variables. Second, not all anomalous events display markedly different energy consumption; small deviations may still be significant when contrasted with data from other consumers within the neighborhood.

In this chapter, we present a novel method for identifying abnormal energy usage events using partially known context information. Our approach recognizes that sensing every possible context variable (such as occupancy or zone temperature) is technically and economically infeasible. Instead, we use context information which is directly available from meter readings: timestamps and metadata attached to the meter identity. Temporal context information, extracted from timestamps, is used for improving the anomaly detection accuracy by picking only the relevant historic data for baseline estimation. This captures the effect of factors such as operating hours (for commercial buildings) and seasonal changes (such as heating/cooling loads) on the consumption characteristics of the buildings.

Neighborhood information (as derived from available metadata) is used for adjusting the anomaly score to account for unknown context variables that influence historically correlated consumption patterns in the same way. For each building or user, the neighborhood is defined

as the set of all other users that have similar characteristics (function, location or demography), and are therefore likely to react and consume energy in a similar way in response to the external conditions. The neighborhood can be predefined based on prior customer information or can be identified through an analysis of historical energy consumption.

The proposed approach consists of three steps:

1. Split the given meter readings into disjoint sets based on available temporal context information.
2. Within each temporal context, run an anomaly detection algorithm separately on each meter's readings.
3. Adjust the anomaly score for individual meter readings based on available neighborhood information.

We validate the effectiveness of the proposed algorithm for both residential and commercial buildings using energy consumption data for a year, thus capturing seasonal trends as well as shorter-term ones. Through experiments, we show the importance of using temporal context and neighborhood information which significantly improves the anomaly detection accuracy in comparison with an existing anomaly detection method proposed in [67].

## 4.2 Definitions and Assumptions

Electric utilities install smart meters in their consumer's buildings and record the aggregate level electricity usage, typically sampled at a one-hour interval, on a massive scale. Our objective is leveraging such opportunistically available large dataset for detecting abnormal energy usage behavior and provide feedback to the consumers for further action. Note that abnormal energy usage may happen due to several reasons such as 1) a faulty appliance consuming more electricity than usual; 2) an electric component failure leading to high or low power consumption; 3) misusage of devices by humans when not required; or 4) genuine usage (e.g. festival). In this

work, we are primarily targeting usage anomalies, any irregular (high or low) energy usage patterns due to human mistake. e.g., lights are left on during non-working hours. Identification of other anomalies may require retrofitting for collecting fine-grained sensor data and operational context, which is not the focus of this work.

### Anomaly Types

Given that anomaly events can occur at any time, we broadly characterize the common anomalies in real-world buildings into two categories:

- **Single instance anomaly:** Occurrence of an anomaly event on a single instance in the given time series of measurements. An example of single instance anomaly is a sudden increase in the building's energy consumption on a given day. Figure 4.1 shows the examples of single point anomalies (a few days within the black boxes) in Meter 2 and 4 as the lighting systems were not turned off by mistake during non-operational hours, e.g. night.
- **Sequence anomaly:** A set of consecutive anomalous events over a short period of time. One example of such an anomaly would be increased consumption for a given building for an entire week. Figure 4.1 shows examples of sequence anomalies (red boxes) in Meter 1 and 4 as some devices were left on during night hours for few days.

### Temporal context sets

As explained in the previous section, several types of context variables can influence energy consumption for a user or a set of users. In order to avoid using context variables, such as fine-grained occupancy and appliance usage patterns, that can only be measured with expensive sensing infrastructure, we only use temporal context variables in this work. Since energy usage events follow regular temporal patterns, we define *temporal context sets* which split the given meter readings into to $N$ disjoint temporal subsets. Figure 4.1 shows the hourly power consumption of a commercial building complex for a year. It shows several temporal context changeover

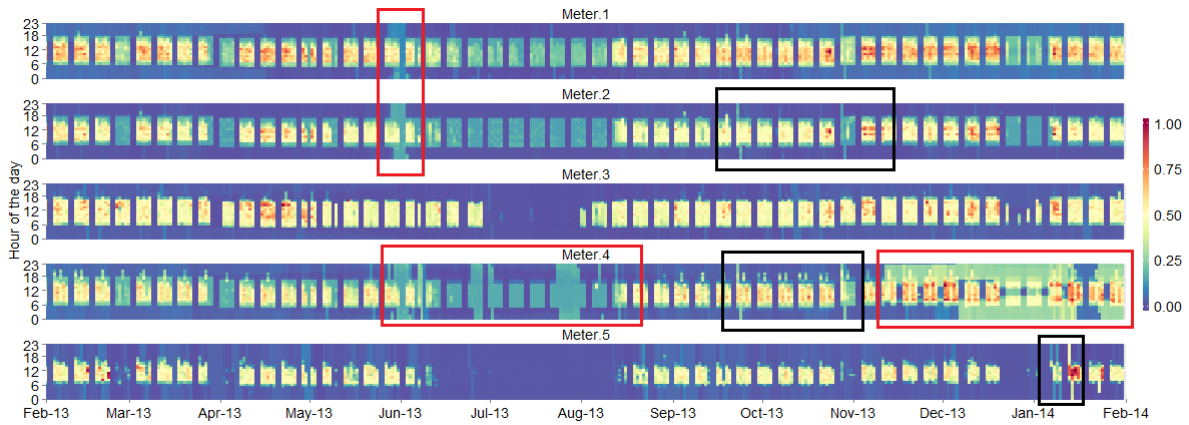



Figure 4.1: Hourly power usage (normalized) of different buildings with in a large commercial building complex (neighborhood) in Sweden for a year from 1st Feb 2013 to 31st Jan 2014. It shows, (a) daily and weekly power usage cycles with seasonal variations during summer, winter and holidays, (b) examples of single point anomaly (marked in black), and (c) examples of sequence anomaly (marked in red). X-axis denotes the day of the year while Y-axis is hour of the day.

events including daily cycle (working and non-working hours), weekly cycle (week days and weekends) and seasonal variations due to summer holidays, weather change and different operational modes of a building. As an example, each meter reading can be classified into one of three temporal context sets, based on the development presented in Section 4.3.1:

- **WorkingDay-BusinessHours:** This set contains meter readings taken during business hours (8 am to 5 pm) on working days (Monday to Friday).
- **WorkingDay-NonBusinessHours:** This set contains meter readings taken outside business hours (5 pm to 8 am) on working days.
- **Weekend:** This set contains meter readings taken during weekends, when commercial premises are unoccupied throughout the day and night, while residential buildings are likely to have increased occupancy.

Multiple temporal context sets can also be defined for each meter reading based on the operating characteristics of a building e.g. residential vs commercial.

**Definition of neighborhood**

The *neighborhood* of a given building (or residence) is defined as the set of buildings (or residences) that are expected to be influenced similarly by the same context variables (known or unknown). This definition is based on available metadata. For example, the neighborhood of a commercial building may be defined to be the set of all other buildings within the same commercial complex (*administrative* neighbors). Alternatively, the neighborhood of a school building may be defined to be the set of all other school buildings in the same geographical area (*functional* neighbors).

Neighborhood information can be predefined by a domain expert based on prior customer information or can be identified through an analysis of historical energy consumption and available metadata. As an example, authors in [120] proposed a framework for grouping the consumers based on several contextual dimensions such as locations, communities, seasons, weather patterns, and holidays. In this work, we assume that identification of neighborhood information is a prior step of applying the proposed anomaly detection method.

Our objective is to identify potential abnormal energy usage events using aggregate (building level) smart meter data as the input. The *abnormality* is quantified by computing anomaly scores for each time period. The anomaly score computation combines metadata based neighborhood definitions with the historical correlation between each pair of time series, in order to compute the pairwise influence of neighbors on each other's individual anomaly scores. In the absence of any metadata, the default neighborhood is the set of all users, and pairwise influences are computed purely using historical correlation.

## 4.3 Anomaly Detection Algorithm

In this section, we describe the algorithm developed for computing anomaly scores for energy consumption data, and also for flagging potential events of interest. The anomaly score, as it

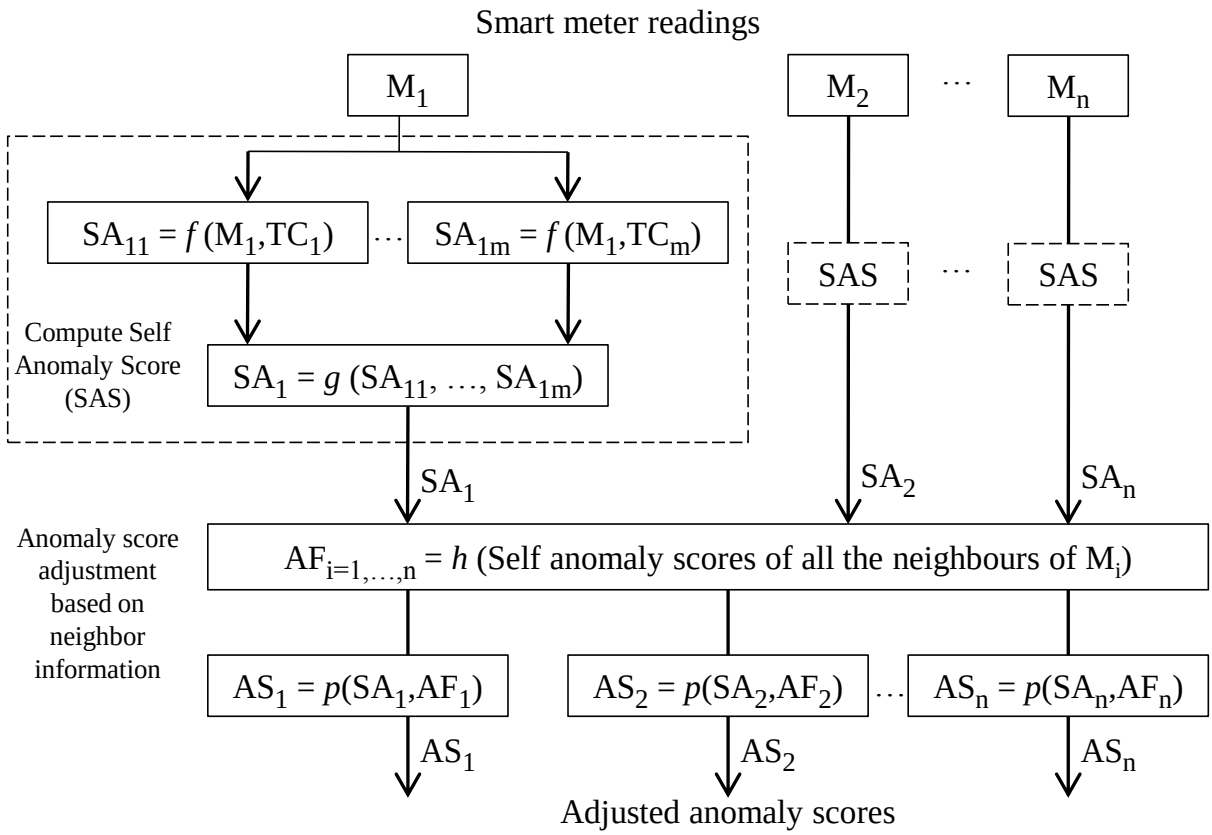



Figure 4.2: Logical flowchart of the proposed anomaly detection algorithm. The function $f$ computes self anomaly score for each meter and for each temporal context set separately followed by function $g$ concatenates them. Function $p$ computes the adjusted anomaly score for each meter data based on the available neighborhood information.

applies to this work, is a scalar in the range $[0, 1]$ (low to high severity) that denotes the likelihood of a given data instance being anomalous. The proposed method consists of three steps for computing this score: (1) split the individual meter readings into disjoint sets based on available temporal context information (Section 4.3.1); (2) apply an anomaly detection algorithm separately on each context set and compute an initial anomaly score (Section 4.3.2); (3) adjust the anomaly score for individual meter readings based on available neighborhood information (Section 4.3.3). Figure 4.2 illustrates the workflow of the proposed anomaly detection method.

### 4.3.1 Splitting data based on temporal context

A suitable period of time in the available data set (for example, the past 60 days) is picked for the analysis described in this section. The proposed algorithm computes anomaly scores for each instance in this time period (for example, for each day in a 60 day period). It can be executed on a rolling basis in order to provide periodic feedback to the end user. The first step in the anomaly detection method involves splitting the given individual meter readings based on the temporal context. As explained in Section 4.2, we use only data timestamps for this classification because they are easily obtainable from existing metering infrastructure. Since human activity typically follows regular temporal patterns, appliance usage is highly correlated to the temporal context. Therefore, computation of anomaly scores based on time-classified data is expected to increase the accuracy of the anomaly detection algorithm.

The specific definitions of temporal context for a physical installation are assumed to be provided by a domain expert. It is assumed that the number of definitions is $C$, and each definition splits the data set into $N$ disjoint subsets. For example, the building manager for a commercial complex could define a classification with $N = 3$ and a periodicity of one week: *WorkingDay-BusinessHours* [Mon-Fri, 8AM-5PM], *WorkingDay-NonBusinessHours* [Mon-Fri, 5PM-8AM], and *Holidays* [Sat-Sun]. In the absence of temporal context information, it can be obtained automatically by applying existing change point detection methods [77, 116]. The algorithm that we propose thus supports the definition of multiple temporal context sets for the same set of data. For example, the same date/time instance could be classified by the time of the day, day of the week, or season of the year. The self-anomaly detection algorithm described in Section 4.3.2 can be run on each temporal context set separately, and the scores can be merged later.

### 4.3.2 Self-anomaly score computation

After time series data from each meter has been classified according to a given temporal context definition $c \in \{1, \ldots, C\}$ into $N$ subsets, each subset is processed independently by an anomaly

**Algorithm 1:** Self anomaly detection algorithm

**Input:** $X^M_{N,L}$: A multivariate time series spanning $D$ days (split into $N$ temporal context sets with $L$ slots each) and $M$ meters

**Output:** $A^m_{n,l}$: Self anomaly score for each time slot

Compute dissimilarity matrix $\Delta^m_n$ using DTW function for all pairs of time slots $(x^m_{n,i}, x^m_{n,j})$ within a given temporal class.

Find the optimal number of clusters $P$ in $\Delta^m_n$ using PAM.

Partition $\Delta^m_n$ into clusters $C_p$, $p \in \{1, \ldots, P\}$ using the k-medoid algorithm, compute the population of each cluster and save in $\bar{S}^m_n$.

Compute the Euclidean distance vector $\bar{D}^m_n$ from each time slot $x^m_{n,l}$ to the medoid of each cluster $C_p$

**forall** $x^m_{n,l} \in$ *one temporal class and meter* $m$ **do**

  **forall** $p \in \{1, \ldots, P\}$ **do**

    $\bar{D}^m_n(p)$ = Euclidean distance ($x^m_{n,l}$, Medoid($C_p$))

  $A^{m*}_{n,l} =< \bar{D}^m_n, \bar{S}^m_n >$

Compute the normalized anomaly score for each $x^m_{n,l}$

$$A^m_{n,l} = \frac{A^{m*}_{n,l}}{\max_{i \in tempclass}(A^{m*}_{n,l})}$$

detection algorithm. This initial analysis, referred to as *Self-anomaly detection*, is summarized in Algorithm 1. The input to the algorithm is the time series from one meter and for a single temporal context (e.g. *WorkingDay-BusinessHours*), and the output is an anomaly score for each instance in the time series. Note that the algorithm only requires historical data for this computation and that no neighborhood information has been introduced at this point.

In this work, we denote each series of consecutive meter readings within a single context set as a *power-time cycle* or *time slot*. For example, all meter readings between 0700 and 1800 on a Monday would form one time slot in the *WorkingDay-BusinessHours* set. Let us denote the readings from meter index $m \in \{1, \ldots, M\}$ in a given neighborhood, and within a given time slot, by $x^m_{(n,l,t)}$, where $n \in \{1, \ldots, N\}$ is the index of the context set, $l \in \{1, \ldots, L\}$ is the index of the time slot, and $t \in \{1, \ldots, T\}$ is the time index (minute or hour of the day) within the time slot. We refer to the collective set of all measurements in a single slot $x^m_{n,l}$, to all the time slots within a single context set collectively by the notation $X^M_{n,L}$, and to the entire data set by the

notation $X^M_{N,L}$. In this work, we use a classification scheme with $N = 3$ and a single temporal context definition ($C = 1$), as described in Section 4.2.

The algorithm computes the dissimilarity matrix $\Delta^m_n$ for all pairs of time slots $(x^m_{n,i}, x^m_{n,j})$, where $i, j \in \{1, \ldots, L\}$ that belong to the same meter and the same temporal class. In order to measure the similarity/dissimilarity between two time slots, we use the *Dynamic Time Warping (DTW)* method [103]. This is a common method for finding the similarity between two time series while accounting for small differences in temporal characteristics. This method is especially useful in the current instance because it compensates for differences in the working hours for two buildings, and also for the shifts caused by daylight saving time (where applicable).

The next logical step is to execute a clustering algorithm on the computed dissimilarity matrix $\Delta^m_n$, thus assigning similar time slots to the same clusters. An unsupervised *k-medoid* clustering algorithm based on Partitioning Around Medoids (PAM) [96] is used for this purpose. The PAM method identifies the optimal number $P$ of clusters in the dissimilarity matrix $\Delta^m_n$, and the k-medoid algorithm subsequently populates the clusters. In contrast with other distance-based clustering algorithms such as k-means, k-medoid algorithm is robust to noise and outliers. Note that the existence of a cluster for a certain set of operating characteristics only implies that these characteristics were observed a significant number of times in the data set. However, the cluster itself may represent an undesirable operating condition from the perspective of a building operator. This issue is discussed further in Section 4.5.

An anomaly score is assigned to each time slot $x^m_{n,l}$ based on the Euclidean distance between this time slot and the medoids of all the clusters, as described in Algorithm 1. The set of Euclidean distances between time slots $x^m_{n,l}$ and each medoid is stored in an $P$-sized vector $\bar{D}^m_n$. The size (population) of each cluster $C_p$, $p \in \{1, \ldots, P\}$ is stored in a $P$-sized vector $\bar{S^m_n}$. Note that $\bar{S}^m_n$ is common to all instances belonging to the same temporal class for a given meter $m$. The unnormalized anomaly score $A^{m*}_{n,l}$ is then derived by the dot product between $\bar{D}^m_n$ and $\bar{S}^m_n$. In order to compute the relative severity of each anomaly, the final step in the self-anomaly detection algorithm is to normalize by the maximum observed value of $A^{m*}_{n,l}$ within the same temporal

**Algorithm 2:** Integrated anomaly detection algorithm with neighborhood comparison

**Input:** $X_{N,L}^{M}$: A multivariate time series spanning $D$ days (split into $N$ temporal context sets with $L$ slots each) and $M$ meters
$A_{n,l}^{m}$: Self anomaly scores for each instance in $X_{N,L}^{M}$

**Output:** $\widehat{A}_{n,l}^{m}$: neighborhood-adjusted anomaly score for each instance in $X_{N,L}^{M}$

$A_{n,l}^{m} = \text{COMPUTESELFANOMALYSCORE}(x_{n,l}^{m})$.

Compute the correlation matrix $\mathbb{C}_n$ of size $M \times L$ between the time series $\{x_{n,1}^{m_1}, \ldots, x_{n,L}^{m_1}\}$ and $\{x_{n,1}^{m_2}, \ldots, x_{n,L}^{m_2}\}$ for each pair of meters $(m_1, m_2)$.

Adjust self-anomaly score based on neighborhood

**forall** $l \in \{1, \ldots, L\}$ **do**

  **forall** $m \in \{1, \ldots, M\}$ **do**

    $\delta_{n,l}^{m} = \sum_{k \neq m} \mathbb{C}_n(k, l) A_{n,l}^{k}$

    $\widehat{A}_{n,l}^{m} = |A_{n,l}^{m} - w \times \delta_{n,l}^{m}|$

class for a given meter $m$. If there are multiple context definitions ($C > 1$), the anomaly scores from each one can be combined to give the collated self-anomaly score $A_{n,l}^{m}$. The final value of $A_{n,l}^{m}$ is fed to the neighborhood adjustment algorithm described in the next subsection.

### 4.3.3 Neighborhood based adjustment

After computing the initial (self) anomaly scores for each time slot $x_{n,l}^{m}$ from individual meters $m$, the final step in the algorithm adjusts these scores based on contemporary self-anomaly scores within the neighborhood of $m$. This adjustment accounts for those unknown contextual factors that influence energy consumption for all meters within a neighborhood in a similar way. An adjustment $\delta_{n,l}^{m}$ for each instance $x_{n,l}^{m}$ is calculated based on the original self-anomaly score $A_{n,l}^{m}$ and the baseline correlation of power consumption between individual members of a neighborhood. The baseline correlation itself is computed using historical comparisons between each pair of meters. As described in Algorithm 2, $\delta_{n,l}^{m}$ is given by the correlation-weighted anomaly scores from the same time slot $(n, l)$, for all meters within the neighborhood of $m$. The neighborhood adjusted anomaly score $\widehat{A}_{n,l}^{m}$ is defined to be the absolute value of the difference between the self-anomaly score $A_{n,l}^{m}$ and the weighted adjustment factor $w \times \delta_{n,l}^{m}$. The parameter $w$ is chosen

between 0 and 1 which decides how much importance to be for the neighborhood for seasonal adjustments. The optimal value for $w$ can be chosen by a domain expert or calculated empirically, as discussed in Section 4.5.

Intuitively, information from other meters within the neighborhood should help the anomaly detection algorithm to differentiate between the effects of unknown contextual factors and anomalous behavior for a single meter. Unknown contextual factors are expected to produce an effect on several meters within the neighborhood. Therefore, the severity of an observed anomaly should be reduced if multiple other meters also report high self-anomaly scores. On the other hand, the anomalous behavior observed for a given meter but not for other meters within the neighborhood (or vice versa) should result in high anomaly scores. Finally, low self-anomaly scores throughout the neighborhood should result in low anomaly scores overall. We note that the definition of $\widehat{A}^m_{n,l}$ given in Algorithm 2 satisfies all of these requirements.

## 4.4 Datasets

We evaluate the proposed anomaly detection method using energy meter readings collected from two different geographical locations. The dataset was collected from commercial and residential buildings located in Sweden and India respectively. Thus, these two data sets represent energy meter readings collected from buildings with different operational characteristics along with influence from diverse context factors such as weather conditions.

### 4.4.1 Commercial buildings

The commercial building data set used for our experiments was collected from a public school campus in Sweden. The school consists of 10 buildings for classrooms and office spaces. These buildings are operated on fairly regular schedules with fixed working days, holidays and daily fixed hours work cycle. The aggregated energy usage of each building within the campus was

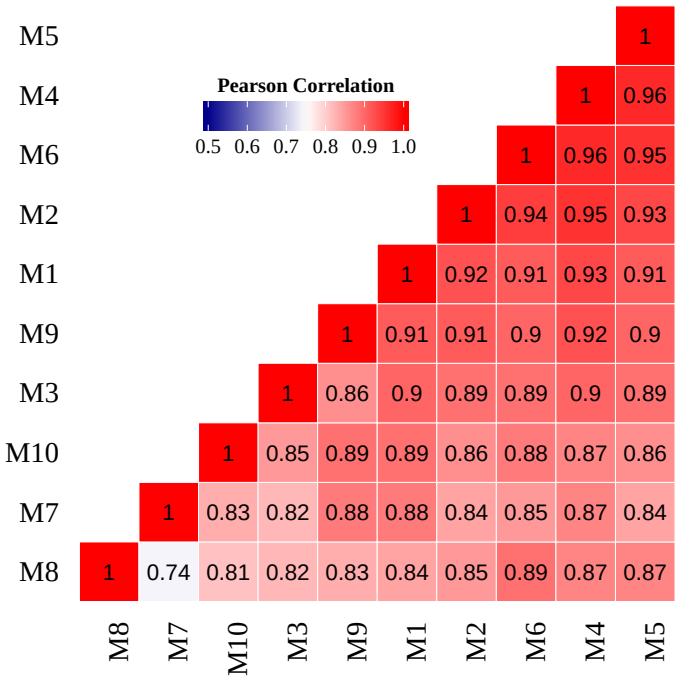



Figure 4.3: The baseline correlation between 10 buildings for a year in the Sweden commercial building data set. Meters are arranged using hierarchical clustering algorithm.

measured separately by smart energy meters. Meters were installed by a third party energy data analytics company for real-time energy monitoring. Aggregated meter readings were sampled at 1 minute to 15 minutes interval and stored in a cloud-based data collection system for more than a year. Figure 4.3 shows the baseline correlation among these buildings for a year.

### 4.4.2 Residential buildings

This data set contains meter readings from a multi-floor residential complex in India. This building complex consists of 8 floors with 3 apartments in each floor, with a total of 24 apartments in a single building. Most residents in these apartments are working professionals who follow regular office hours. Hence the apartments are typically not occupied during daytime but display high activity during morning and evening. These apartments are equipped with common home appliances such as lighting systems, refrigerator, heaters and air-conditioners. A smart meter was

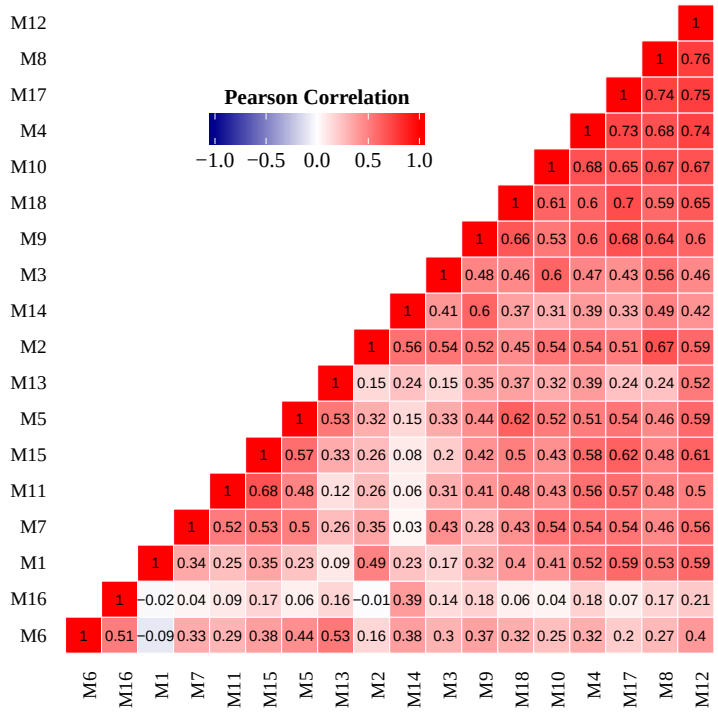



Figure 4.4: The baseline correlation between 18 apartments for a year in the Indian residential buildings dataset. Meters are arranged using hierarchical clustering algorithm.

installed for each apartment separately for energy monitoring. Energy meter readings are sampled at 30 seconds interval and stored in an open-source meter data aggregation system. Among the 24 apartments, we selected 18 for our analysis because there was a lot missing readings for the remaining 6 apartments. Similar to commercial building dataset, we use data spanning one year for our experiments, between August 2013 and July 2014, to account for the seasonal variations. Figure 4.4 shows the baseline correlation among these apartments for a year.

### 4.4.3 Preprocessing and anomaly injection

The two data sets represent smart meter readings collected from different geographical locations, weather conditions, daily/weekly usage cycles. We down sampled the meter readings to 1-hour resolution (averaged over 1-hour window) for each data set. Because utility companies generally collect meter readings as low as 1-hour resolution and for a valid comparison of the proposed

method with an existing method (See Section 4.5.1) which also used 1-hour resolution data. Further, days which contained more than 10% of missing values were excluded from our analysis. For the rest of the missing data (1% for commercial and 15% for residential data), the missing values were replaced by a weighted average of the rest of the day.

**Verification procedure for anomalies**

Since the analysis presented in this work is based on real historical data, ground truth information about actual anomalies is not readily available. Discussions with the owners of the commercial data set were used to confirm the veracity of certain anomalies in the Swedish data. These confirmed cases are presented in the next section. For residential data, our analysis is restricted to identifying a particular abnormal energy usage event where one or more appliances are operational outside usual hours. An example of such an anomaly would be the operation of heating/cooling loads outside office hours in commercial spaces.

In summary, ground truth comparison in the Swedish data set was carried out through discussion of suspicious events (shown in Figure 4.1) with the facility managers. For the Indian residential building data set, we surveyed the apartment owners and collected the details about all the appliances being used and their power ratings. We manually matched the energy usage of some of the high-power consuming appliances with changes in the raw power meter readings. This exercise was used to model the energy ratings of the appliances. Subsequently, we injected the signatures of 'anomalous usage' of the appliances (left on for few hours to few days) into the raw power readings and marked it as ground truth for our analysis. Table 4.1 summarizes the details about the injected anomaly events. As we use three context sets with a minimum time interval of 12 hours (Section 4.3.1), the required minimum duration of a single anomaly should be 12 hours. To cover the worst case scenario, which is below 12 hours, we injected single anomaly for 6-24 hours. As sequence anomaly occurs in consecutive time slots, we injected the sequence anomaly for 2-3 days (required minimum no of days is two).

Table 4.1: Details about the injected abnormal energy usage events into the residential building dataset.

| Appliance | Power (kW) | Anomaly type | No. of instances | Duration |
|---|---|---|---|---|
| Air conditioner | 1.8 | single | 5 | 6 - 24 hours |
| Air conditioner | 2.0 | sequence | 2 | 2 - 3 days |
| Room heater | 2.2 | single | 4 | 6 - 24 hours |
| Room heater | 2.2 | sequence | 3 | 2 - 3 days |

## 4.5 Experimental Results

One of the major challenges with evaluating anomaly detection methods is the unavailability of fine-grained ground truth data about the actual anomalies from real-world buildings. Hence we adopt a similar evaluation method presented in [88], and analyze the computed anomaly score case by case for known abnormal energy usage events, such as lights are not turned off during non-working hours. Further, we employed a visual analytics method, similar to [94], for visualizing the actual data and anomaly score in a convenient manner, and to highlight the potential anomalies to the building owners. Therefore, our experimental results are focusing on analyzing the signature of some known anomalies with respect to the temporal context sets and neighborhood information.

### 4.5.1 Baseline Methods

We compare the performance of the proposed anomaly detection method with an existing algorithm described for commercial building. Additionally, we also compare the performance of the integrated algorithm with two simpler versions that use only self-anomaly detection method.

**Self Anomaly - No Context (SANC)**: This is the self-anomaly detection method described in Section 4.3.2 but without using any temporal context information. Energy meter readings were directly fed into Algorithm (1) without splitting them into temporal context sets. We selected this method to show the significance of using available temporal context information for improving the anomaly scores for known anomaly events.

**Self Anomaly, but using Temporal Context (SATC)**: This is also the self-anomaly detection method described in Section 4.3.2, but now using all the available temporal context information described in Section 4.3.1. However, it does not adjust the computed anomaly score using available neighborhood information. We selected this method to show the significance of using available neighborhood information for adjusting the anomaly score to factor the influence of unknown contexts for a year.

**Self Anomaly - HP (SAHP)**: This anomaly detection method was proposed by Bellala et al from HP for identifying daily anomalous events for commercial buildings [67]. We chose this method because it shares a similar idea with the self-anomaly method but without using any temporal or neighborhood information.

The baseline and the proposed anomaly detection algorithms were implemented in R using its built-in package libraries. Also, we developed a R-Shiny web application and released the code in open-source [40]. We present the identified abnormal energy usage events and compare their performance in the below sections.

### 4.5.2 Analysis of commercial building data

We executed the proposed anomaly detection method on the Sweden commercial building dataset. We used a temporal context set, as discussed in Section 4.3.1, as the building exhibits fixed daily and weekly cycle. A brief discussion with the data owner revealed that all the buildings are from a single administrative neighborhood as the baseline correlation among these buildings is high, as shown in Figure 4.3. Figure 4.5 illustrates the computed anomaly score of the proposed algorithm along with all the baseline methods for a single smart meter. We selected this particular meter data as it contains many instances of single and sequence anomalies (as shown in Figure 4.1) as compared to other meters.

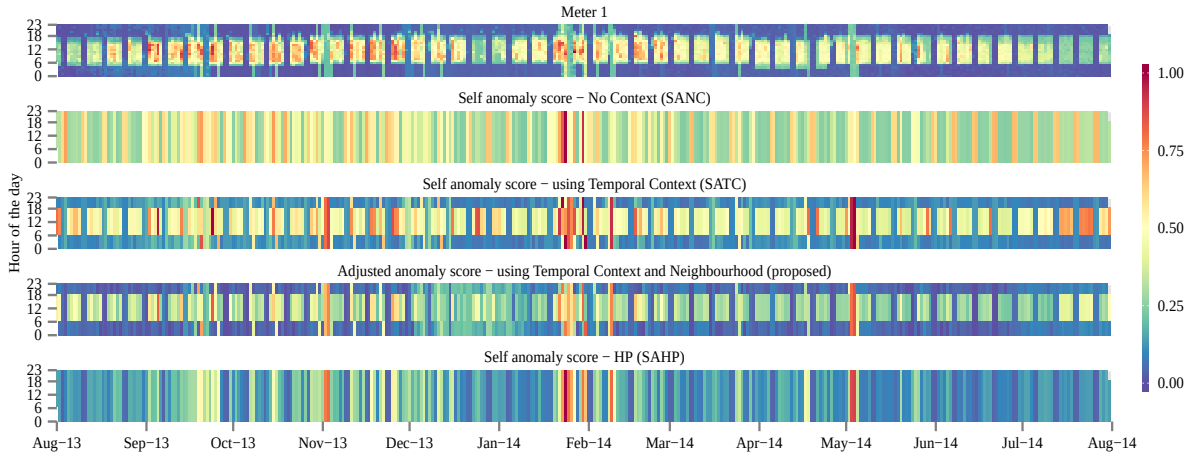



Figure 4.5: Hourly meter readings of a Sweden commercial building with computed anomaly score by different baseline and proposed anomaly detection methods. It shows several instances of point and sequence anomalies and how the computed anomaly score differs using the temporal and neighborhood information.

### Single point anomaly

We consider three instances of noticeable single point anomalies in the raw power readings which occurred on September 25, October 4 and November 14, as shown in Figure 4.5. Both SANC and SAHP were not able to assign a high score for them, as the (total) power usage of those anomalous days, although there was anomalous usage during night hours, is similar to other days. However, SATC is able to assign higher score as it splits the power usage of those days into different context sets, *WorkingDay - BusinessHours* and *WorkingDay - NonBusinessHours*. Using the neighborhood information, proposed algorithm decreases the computed anomaly score as the similar usage was observed in some of the other meters as well.

### Sequence anomaly

We consider three instances of sequence anomalies (abnormal power consumption during weekends) which occurred during November, February, and May, for evaluating the performance of the proposed anomaly detection method. We can observe that, without using any temporal context information, SAHP outperforms SANC as it assigns higher anomaly score for anomalies

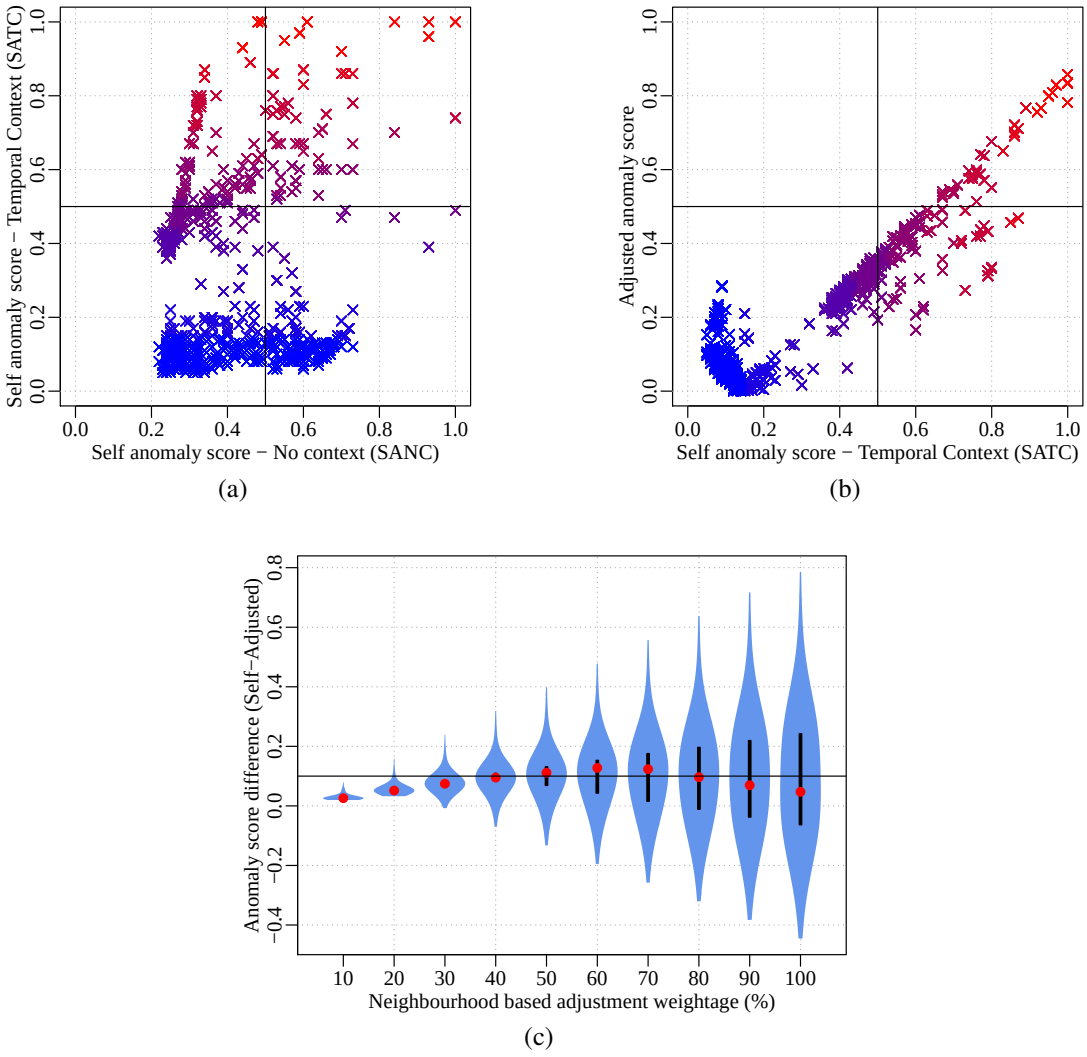



Figure 4.6: Anomaly score comparison of (a) self anomaly score without using any context verses self anomaly score using only the temporal context, (b) self-anomaly score only using the temporal context verses using available neighborhood information, and (c) a violin plot (a combination of box and density plot) shows the differences between anomaly scores (self minus adjusted), by using different adjustment weights for the Sweden commercial building dataset.

happened during November and May. Using the temporal context information, SATC is able to assign a higher score for all the three sequence anomalies. However, after adjustment, it assigns a lower score for the anomaly event that happened in Feb.

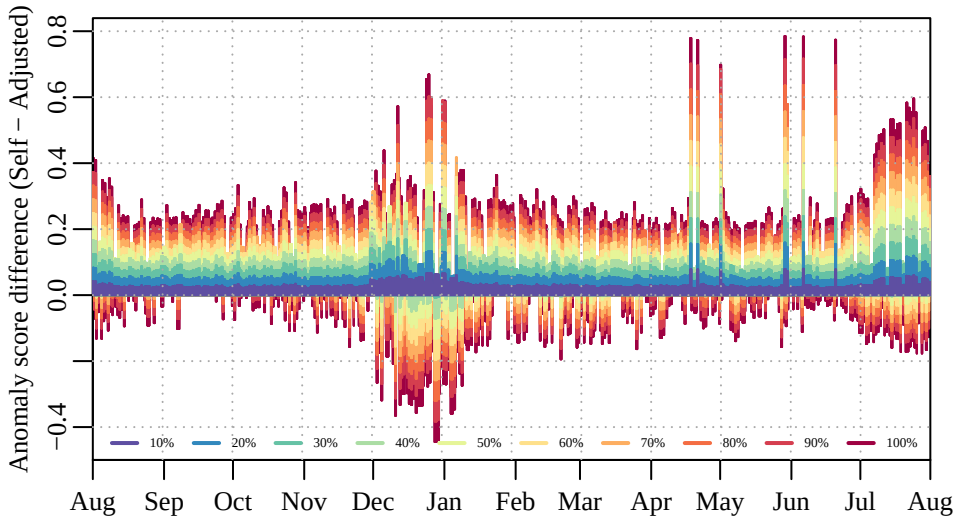



Figure 4.7: Adjusted anomaly score difference for different weights over time for the Swedish commercial building data set. The curve with the smallest magnitude corresponds to a weight of 10% and the one with the highest magnitude corresponds to a weight of 100%. Positive values indicate a reduction in the anomaly score after neighborhood comparison and vice versa.

### Anomaly classification

Figure 4.6a shows a scatter plot of self-anomaly scores with and without using temporal context sets. The spread of the scores indicates the influence of the temporal context sets for adjusting anomaly scores based on seasonal changes. Similarly, Figure 4.6b shows a scatter plot of self-anomaly scores using temporal context and neighborhood adjusted scores, for a single meter in the Swedish commercial complex data. This visualization also highlights the unique characteristics of the proposed algorithm, by automatically classifying different classes of anomalies in the data. The lower left corner of the figure represents nominal operating conditions when both the self-anomaly and adjusted anomaly scores are low. The bottom right quadrant represents points that appeared anomalous to the self-anomaly algorithm, but their severity was downgraded after comparison with neighbors. These instances typically represent events such as festival periods or summer vacations (for schools) and are unlikely to be actual anomalies. Instead, these points denote the contribution of the neighborhood comparison step to the reduction of the number of false positives in the anomaly detection algorithm.

The top left quadrant of the figure shows instances that did not appear anomalous at first, but their severity was upgraded after neighborhood comparison. Complementary to the discussion above, these points represent the contribution of the neighborhood comparison towards reducing the false negative rate in the algorithm. Finally, the top right portion of the figure represents points that were deemed to be highly anomalous, both by the self-anomaly detection and after neighborhood comparison. These points represent the most confidently flagged anomalous instances and should be investigated by human supervisors on a high priority basis.

**Anomaly score adjustment using neighborhood information**

After computing the initial anomaly score, we assigned a different percentage of weights for the neighbors and calculated the adjusted anomaly score. Figure 4.6c illustrates the difference between self and the adjusted anomaly score while using different weights ranging from 10% to 100%. It is observed that up to assigning weights 60% the average difference between the self and adjusted score is increased linearly. After that, it started to decrease for weights higher than 60%, as shown inFigure 4.6c. Figure 4.7 shows how adjustment varies over time for different seasons with different neighborhood weights. We can observe that higher adjustment during January for accounting the seasonal changes. Also, there are some instances of higher adjustment during May to account for the abnormal energy usage events which are also visible in Figure 4.5.

### 4.5.3 Analysis of residential building data

We executed the proposed anomaly detection algorithm over all the 24 smart meter readings in the Indian residential building data set. We used a temporal context set, as discussed in Section 4.3.1, as the building occupants exhibit regular daily and weekly cycle similar to the commercial buildings. Further, all the buildings were from the same neighborhood. Figure 4.8 illustrates the computed anomaly score of the proposed algorithm and the baseline methods for a single smart meter. We injected the anomalies, shown in Table 4.1, into one of apartment me-

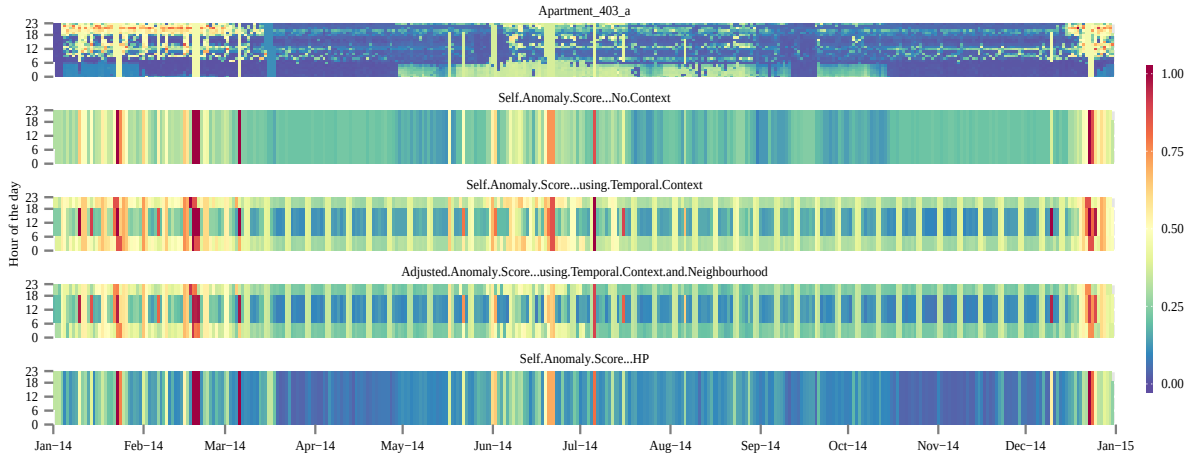



Figure 4.8: Hourly meter readings of an Indian residential building with computed anomaly score by different baseline and proposed anomaly detection methods. It shows several instances of point and sequence anomalies and how the computed anomaly score differs using the temporal and neighborhood information.

ter data. Also, we applied a temporal context set which is similar to the commercial building experiment to account for the daily/weekly energy usage cycle.

### Self anomaly score

In contrast with commercial buildings, variation in the energy usage patterns are high in residential buildings. As shown in Figure 4.4, the baseline correlation between residential apartments was diverse. Specific to India, as shown in Figure 4.8, there is higher power consumption during summer and winter due to the extreme weather conditions. After calculating the anomaly scores, we set a threshold of selecting top 10% of anomalies for the analysis. Among those 14 anomalies that we injected, SATC was able to assign a higher score for all of them, whereas SAHP identified only 12 (missed those single point anomalies happened during July and August).

### Anomaly score adjustment using neighborhood information

Similar to the commercial building, we assigned a different percentage of weights for the neighbors and calculated the adjusted anomaly score. Figure 4.9c plots the difference between self-anomaly score and adjusted using different percent of weights ranging from 10% to 100%. In

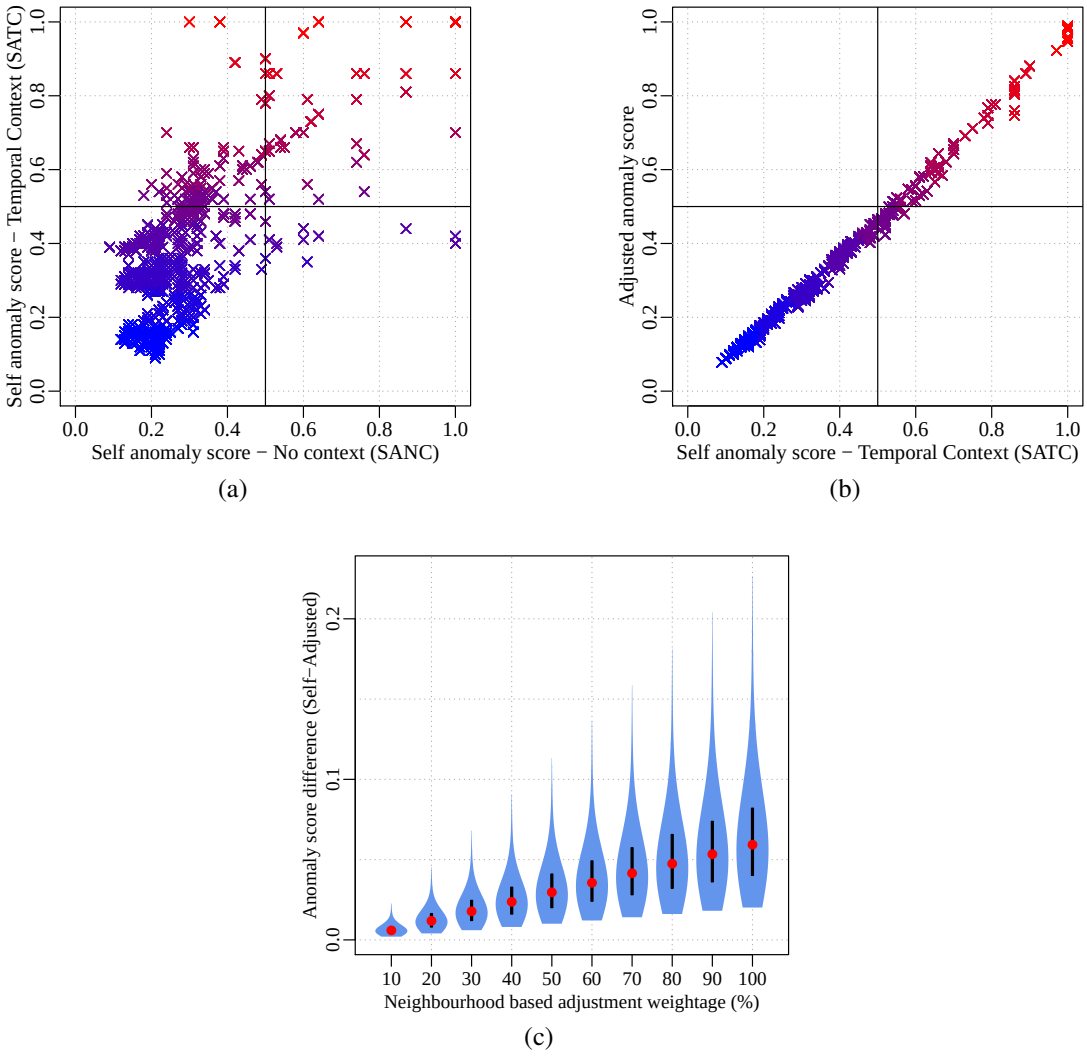



Figure 4.9: Anomaly score comparison of (a) self anomaly score without using any context verses self anomaly score using only the temporal context, (b) self-anomaly score only using the temporal context and using available neighborhood information, and (c) violin plot (a combination of box and density plot) shows the differences between anomaly scores (self minus adjusted), by using different adjustment weights for the Indian residential building dataset.

contrast with commercial buildings, the baseline correlation between the residential apartments is diverse, as shown in Figure 4.4. Due to that the maximum adjustment factor was 0.2. It is observed that the difference between the self and adjusted score is increased linearly with respect to neighborhood weights. Figure 4.10 shows how the adjustment varies over time for different seasons while using different neighborhood weights. We can observe that increase in the adjustment during July to account for the seasonal changes during summer.

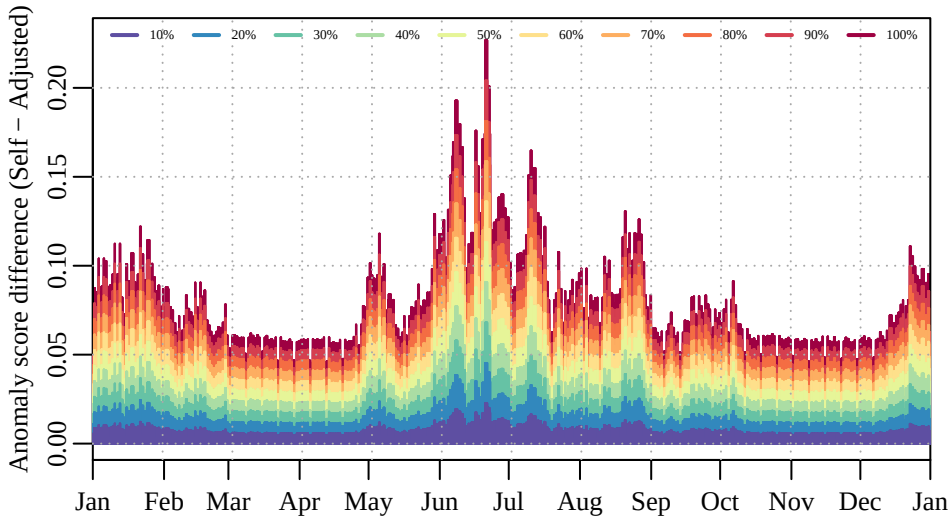



Figure 4.10: Adjusted anomaly score difference for different percentage of weights over time for the Indian residential building data set. The curve with the smallest magnitude corresponds to a weight of 10% and the one with the highest magnitude corresponds to a weight of 100%. Positive values indicate a reduction in the anomaly score after neighborhood comparison.

Similar to the anomaly characterization for the commercial building data set, Figure 4.9a shows a scatter plot of self-anomaly scores with and without using temporal context sets. Similarly, Figure 4.9b plots the self-anomaly score against the adjusted anomaly score which shows the classification of anomaly events. In contrast with the commercial building, the scatter plot looks narrow because of the higher variation in the baseline correlation in residential building users.

## 4.6 Related Work

Several anomaly detection methods for different domains have been proposed in the literature for identifying potential anomalous events [73]. Authors in [114] proposed an unsupervised anomaly detection method for identifying potential abnormal energy usage days. They used robust statistical methods based on the variability of mean and standard deviation in the power usage. Similar to that, authors in [70] proposed a generalized extreme studentized deviate method

based on the variation of mean and standard deviation of the measured data. However, these methods are limited to identify the single point anomaly events.

Authors in [74] proposed a framework for detecting energy usage outliers for smart buildings. They used a suffix tree representation of the energy usage activities for grouping the subsequence energy usage events and cluster them for identifying anomalous events. A similar method but without using clustering algorithm was proposed in [67]. They use a multi-dimensional scaling method for reducing the dimension of the dissimilarity matrix and assign density of a point using the kNN algorithm.

Several multivariate anomaly methods have been proposed in the literature. Authors in [78] presented a graph-based algorithm for detecting and characterizing the anomalies. Since many real-world events are interrelated, Granger causality methods have also been used in the literature for time series anomaly detection [109]. However, all these methods do not account the context factors which influence the occurrence of anomaly events. An anomaly detection framework for monitoring the Hadoop Map-reduce tasks using system-level context information was proposed in [91]. Our proposed self-anomaly detection algorithm shares similar idea with them for using the context information. However, the adjustments of self-anomaly scores across the meters in a neighborhood are essential for energy domain for accounting the seasonal changes.

In contrast with all the existing methods, the proposed method uses the context information which is collected as part of the meter data collection system. Further, we evaluate the proposed method using a year-long data set from both commercial and residential buildings.

## 4.7 Summary

In this chapter, we described an anomaly detection methodology for energy consumption data time series, that used information from neighboring meters to qualify its output. The neighborhood for a meter (the source of each time series) could be defined a prior or could be identified directly from the data. We showed that incorporating such information was an effective safeguard against the identification of spurious anomalies (false positives), as well as against the omission of real anomalies (false negatives). The generic nature of the algorithm ensures that it is effective for a wide-range of applications, including residential and commercial complexes.

# Chapter 5

# Conclusions and Future Work

Buildings are one of the largest energy consumers around the world, accounting for over 40% of energy usage both in developed and developing countries. They are increasingly instrumented with novel Cyber-Physical control systems that use heterogeneous sensors and actuators for monitoring, automating, and optimizing different building operations for improving energy efficiency. Several efforts have been sought in the past decade by both research community and several sectors of the government and industry to reduce the energy footprint of buildings. In this thesis, we described the design, development, deployment and experimental validation of software systems and analytical methods towards optimizing the energy usage in buildings. We implemented several real-world applications for identifying abnormal energy usage events ranging from rule-based methods to complex context-aware analytics and scaling them to a network of buildings. Further, the software systems and analytical methods presented in this thesis were released in open-source for community use [29, 40, 42].

## 5.1 Summary of contributions

This thesis makes following contributions to the advancement of making buildings more energy efficient.

1. **Decentralized energy management system for buildings**: We proposed a decentralized and extensible middleware system architecture, called *SensorAct*, for energy management in buildings. It provides flexible interfaces, in the form RESTful APIs, for integrating the underlying heterogeneous sensing and control systems into a single platform for better management. To ease the development of third-party applications and to extend existing features, *SensorAct* architecture provides a scripting framework for automating the various energy management functions. It also supports a rule-based mechanism for fine-grained sensor data and control sharing for external applications and occupants. Further, the *SensorAct* system provides programming abstractions for developers, through RESTful APIs and *Tasklets*, for accessing the underlying networked sensing systems and processing their data, enabling the development of extensible energy monitoring applications.

2. **Reusable sensor data analytics service for identifying the operational context of the building**: We proposed a reusable middleware system service, called *OpenBAN*, for analyzing the sensor data to infer the operational context of the building, which enables the identification of complex context-based abnormal energy usage events. *OpenBAN* service consists of a runtime environment for developing and scheduling a pipeline of processing elements, called as *Contextlet*, for sensor data analytics. As a service, it can be integrated with other building management systems for providing extensible and scalable sensor analytics support. In our prototype, we integrated the *OpenBAN* service with *SensorAct* system and developed a number of energy monitoring applications, such as energy disaggregation and sprinkler usage violation, to show their combined utility.

3. **Scalable anomaly detection method using readily available sensor data for a network of buildings**: We proposed a novel unsupervised method for identifying abnormal energy usage events for a large number of buildings within a neighborhood, using only the smart energy meter readings. The proposed approach recognized that sensing every possible

context variables which affect energy usages, such as occupancy and indoor climate, is technically and economically infeasible. So we used only the readily available context information which is directly available from meter readings and utilities, making the approach scalable for a large number of buildings. We validated the effectiveness of the proposed method using real-world smart meter readings for both commercial and residential buildings across two different geographical regions. We showed that the usage of readily available context information improves the anomaly detection accuracy, and it outperforms a baseline method proposed in the literature.

## 5.2 Future directions

In this section, we list out the limitations of software systems and analytical methods described in this thesis and suggest some ideas for future work.

### 5.2.1 Vendor agnostic integration and portable building applications

The underlying sensing and control systems in the buildings are built with vendor-specific communication protocols, access methods, and configurations. One of the challenges is mapping these heterogeneous resources into a common namespace for uniform access across the building applications. Even though these systems use some standardized communication protocols, such as BACnet and Modbus, there is no standard way to configure them as different engineers use different naming convention and metadata schema.

The *SensorAct* system described in Chapter 2 relies on developing separate Gateways to integrate a specific sensing and control system, which requires tedious manual work. One approach could be developing a vendor agnostic gateway which will automatically map the underlying naming conventions into a common namespace which can be used by the top-layer applications. Such a system will enable the development of a truly portable building applications which can

run on any building with minimal or no modification to the application code. There are some recent research works [69] that attempt to address these challenges but still limited to specific settings, e.g. HVAC. Another approach could be to use an ontology-based model by creating a knowledge repository of the building subsystem components.

### 5.2.2 Writing secure and fault-tolerant building applications

The building energy management applications often need to condition the ambience of various building regions, e.g., turning on or off the lighting systems based on occupancy level. So they should be secure and fault-tolerant to the failures in building subsystems to avoid any adverse effects. The scripting framework described in the *SensorAct* system provides an execution environment for running the building applications in a sandbox. However, it assumes that the application developer will write the code which does not create any adverse effects or race-conditions, particularly when controlling the actuators. This can be mitigated by providing an automatic mechanism for verifying the application code before actual deployment. The identified adverse effects, if any, could be reported to the building administrator for further investigation. Several formal program verification methods have been proposed in the literature which can be employed for solving this problem.

### 5.2.3 Usability study

We have evaluated the software systems presented in this thesis based on real-world deployment and experimental validation of real-world energy management applications, as a proof of concept. We have also analysed the system performance to identify how much load the system can handle for practical deployments. The usability evaluation of the system could be an interesting future direction to identify how well users can learn and use the system. It can be targeted towards different stakeholders of the system, namely application developers, building administrators, and end users.

We can conduct a usability study to identify how easy for the developers to write the building applications on top of the proposed scripting framework in *SensorAct*and analytics engine in *OpenBAN*, in contrast with existing legacy systems. Another interesting study could identify how easy and useful the system is for the building administrator for performing various day-to-day tasks. Finally, after deploying the energy management applications, we can conduct a survey among the building occupants to identify posterior effects such as how they like and use the applications, any behavioral changes in saving energy, and any inconveniences to them.

# Bibliography


[1] Procedures for Commercial Building Energy Audits — American Society of Heating, Refrigerating, and Air-Conditioning Engineers. https://www.ashrae.org/resources--publications/bookstore/procedures-for-commercial-building-energy-audits. [Online; accessed September 2016]. 1.1

[2] BACnet - A Data Communication Protocol for Building Automation and Control Networks. http://www.bacnet.org. [Online; accessed September 2016]. 2.1

[3] PG&E Pilot Yields 7.7% Energy Savings. http://www.bidgely.com/blog/pge-pilot-yields-7-7-energy-savings/. [Online; accessed September 2016]. 1.1

[4] BigML. http://bigml.com. [Online; accessed September 2016]. 3.6

[5] Blower door testing. https://en.wikipedia.org/wiki/Blower_door. [Online; accessed September 2016]. 1.1

[6] Energy Savings Potential of Solid-State Lighting in General Illumination Applications. http://www.nlb.org/index.cfm?cdid=10972&pid=10225. [Online; accessed September 2016]. 1.1

[7] DOE, Buildings Energy Data Book, Table 3.1.5. http://buildingsdatabook.eren.doe.gov/ChapterIntro3.aspx. [Online; accessed September 2016]. (document), 1.2

[8] CurrentCost - EnviR. http://www.currentcost.com/product-envir.html. [Online; accessed September 2016]. 1.1

[9] Power and Energy Meters - Eaton. http://www.eaton.in/Eaton/ProductsServices/Electrical/ProductsandServices/PowerQualityandMonitoring/PowerandEnergyMeters/index.htm. [Online; accessed September 2016]. 2.4.1

[10] Short-Term Energy Outlook - U.S. Energy Information Administration (EIA). https://www.eia.gov/forecasts/steo/report/renew_co2.cfm. [Online; accessed September 2016]. 1.1

[11] Energy audit. https://en.wikipedia.org/wiki/Energy_audit, . [Online; accessed September 2016]. 1.1

[12] Energy demand management. https://en.wikipedia.org/wiki/Energy_demand_management, . [Online; accessed September 2016]. 1.1

[13] Energy Flow Charts: Charting the Complex Relationships among Energy, Water, and Carbon. https://flowcharts.llnl.gov/, . [Online; accessed September 2016]. (document), 1.1, 1.1

[14] ENERGY STAR — The Simple Choice for Energy Efficiency. https://www.energystar.gov/, . [Online; accessed September 2016]. 1.1

[15] Esper: Event Processing for Java. http://www.espertech.com/products/esper.php. [Online; accessed September 2016]. 3.6

[16] FireSense: Firewall-Based Occupancy Sensing. https://github.com/nesl/FireSense. [Online; accessed September 2016]. 2.5.2

[17] Flyport system on module platform. http://www.openpicus.com/site/products. [Online; accessed September 2016]. 2.4.1

[18] Green Button Data. http://www.greenbuttondata.org. [Online; accessed September 2016]. 3.2.1

[19] HomeSeet Inc. http://www.homeseer.com. [Online; accessed September 2016]. 2.5.2

[20] Honeywell Building Solutions — BMS — Commercial Buildings. https://buildingsolutions.honeywell.com/. [Online; accessed September 2016]. 1.2, 2.6.1

[21] 2012 International Energy Conservation Code. http://publicecodes.cyberregs.com/icod/iecc/2012/icod_iecc_2012_cover.htm. [Online; accessed September 2016]. 1.1

[22] IFTTT. https://ifttt.com. [Online; accessed September 2016]. 3.6

[23] Building Management Systems - BMS — Johnson Controls. http://www.johnsoncontrols.com/buildings/building-management. [Online; accessed September 2016]. 1.2, 2.6.1

[24] Jython. http://www.jython.org. [Online; accessed September 2016]. 2.4.2

[25] LabSense: An Extensible and Easily Configurable Energy Monitoring System. http://nesl.ee.ucla.edu/document/show/436. [Online; accessed September 2016]. 2.2.1, 2.4.1, 2.5.2

[26] Lua. http://www.lua.org. [Online; accessed September 2016]. 2.4.2

[27] The Modbus Organization. www.modbus.org. [Online; accessed September 2016]. 2.2.1

[28] SensorActuatorManager. https://github.com/nesl/SensorActuatorManager. [Online; accessed September 2016]. 3.4.2

[29] OpenBAN. https://github.com/nesl/OpenBAN, . [Online; accessed September 2016]. 3.3, 5

[30] OpenCPU. https://opencpu.org, . [Online; accessed September 2016]. 3.3.3

[31] OpenCPU - Public Server. https://www.opencpu.org/demo.html, . [Online; accessed September 2016]. 3.5

[32] OpenML. http://openml.org, . [Online; accessed September 2016]. 3.6

[33] OpenPy. https://github.com/game-time/OpenPy, . [Online; accessed September 2016]. 3.3.3

[34] pfSense - World's Most Popular Open Source Firewall. http://www.pfsense.org. [Online; accessed September 2016]. 3.4.3

[35] Play Framework - Build Modern and Scalable Web Apps with Java and Scala. https://www.playframework.com/. [Online; accessed September 2016]. 3.3

[36] Quartz Scheduler. http://quartz-scheduler.org. [Online; accessed September 2016]. 2.4.2, 3.3

[37] Intelligent Rack Power Distribution Units — Power Management — Raritan. http://www.raritan.com/products/power-distribution. [Online; accessed September 2016]. 2.4.1

[38] Raspberry Pi. https://www.raspberrypi.org. [Online; accessed September 2016]. 2.4.2

[39] DOE, Buildings Energy Data Book, Table 2.1.5. http://buildingsdatabook.eren.doe.gov/ChapterIntro2.aspx. [Online; accessed September 2016]. (document), 1.2

[40] Anomaly Detection Method for Smart meters. https://github.com/pandarasamy/anomaly_detection. [Online; accessed September 2016]. 4.5.1, 5

[41] SEAI - Running an Energy Awareness Campaign. http://www.seai.ie/EnergyMAP/Energy_Awareness/Implement_your_campaign/Running_an_Energy_Awareness_Campaign.html. [Online; accessed September 2016].

1.1

[42] SensorAct VPDS v2.0. https://github.com/iiitd-ucla-pc3/SensorActVPDS-2.0. [Online; accessed September 2016]. 2.4.2, 5

[43] Siemens Building Technologies. https://www.buildingtechnologies.siemens.com. [Online; accessed September 2016]. 1.2, 2.6.1

[44] Smappee Offers World's First Itemized Electricity Bill - Smappee. http://www.smappee.com/us/blog/press-en-2015-03-11/. [Online; accessed September 2016]. 1.1

[45] Building Automation Systems — Controls — Trane Commercial. http://www.trane.com/commercial/north-america/us/en/markets/data-centers/controls.html. [Online; accessed September 2016]. 1.2, 2.6.1, 3.6

[46] Vera - Smarter Home Control. http://getvera.com, . [Online; accessed September 2016]. 3.1.1

[47] Power/Energy Monitoring - Veris Industries. www.veris.com/category/power-fslenergy-spcmonitoring.aspx, . [Online; accessed September 2016]. 2.4.1

[48] Wemo Home Automation. http://www.belkin.com/us/wemo. [Online; accessed September 2016]. 3.6

[49] Wolfram Alpha. http://www.wolframalpha.com. [Online; accessed September 2016]. 3.6

[50] Buildings Energy Data Book. *Energy Efficiency and Renewable Energy Department*, 2011. 1.1

[51] Mango M2M. http://mango.serotoninsoftware.com, 2016. [Online; accessed September 2016]. 2.6.3, 3.6

[52] Nimbits. http://www.nimbits.com, 2016. [Online; accessed September 2016]. 2.6.3

[53] Sen.se. http://open.sen.se/, 2016. [Online; accessed September 2016]. 2.6.3, 3.6

[54] Simple Network Management Protocol. https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol, 2016. [Online; accessed September 2016]. 2.4.1

[55] Xively Public Cloud for the Internet of Things. http://xively.com, 2016. [Online; accessed September 2016]. 2.6.3, 3.2.1, 3.3.1, 3.6

[56] Kaa. https://www.kaaproject.org/, 2018. [Online; accessed January 2018]. 2.6.3

[57] openHAB . https://www.openhab.org/, 2018. [Online; accessed January 2018]. 2.6.3

[58] SmartThings. https://www.smartthings.com/, 2018. [Online; accessed January 2018]. 2.6.3

[59] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Global sensor networks. *EPFL, Lausanne, Tech. Rep*, 2006. 2.6.2

[60] AEO. US Energy Information Administration. *AEO2011: Annual Energy Outlook*, April 2011. 1.1

[61] 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*, pages 201–202. ACM, 2012. 1.1, 1.2, 3.1.2

[62] Yuvraj Agarwal, Thomas Weng, and Rajesh K Gupta. The energy dashboard: improving the visibility of energy consumption at a campus-wide scale. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 55–60. ACM, 2009. 1.1

[63] Yuvraj Agarwal, Rajesh Gupta, Daisuke Komaki, and Thomas Weng. Buildingdepot: an extensible and distributed architecture for building data storage, access and sharing. In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 64–71. ACM, 2012. 2.6.2

[64] Pandarasamy Arjunan, Nipun Batra, Haksoo Choi, Amarjeet Singh, Pushpendra Singh, and Mani 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*, pages 80–87. ACM, 2012. 2.3.1

[65] Nipun Batra, Jack Kelly, Oliver Parson, Haimonti Dutta, William Knottenbelt, Alex Rogers, Amarjeet Singh, and Mani Srivastava. Nilmtk: An open source toolkit for non-intrusive load monitoring. In *Proceedings of the 5th international conference on Future energy systems*, pages 265–276. ACM, 2014. 1.2, 3.4.1

[66] Christian Beckel, Leyna Sadamori, and Silvia Santini. Automatic socio-economic classification of households using electricity consumption data. In *Proceedings of the fourth international conference on Future energy systems*, pages 75–86. ACM, 2013. 3.2.2, 3.6

[67] Gowtham Bellala, Manish Marwah, Martin Arlitt, Geoff Lyon, and Cullen E Bash. Towards an understanding of campus-scale power consumption. In *Proceedings of the Third ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 73–78. ACM, 2011. 1.1, 1, 4.1, 4.1, 4.5.1, 4.6

[68] Gowtham Bellala, Manish Marwah, Martin Arlitt, Geoff Lyon, and Cullen Bash. Following the electrons: methods for power management in commercial buildings. In *Proceedings of the 18th international conference on Knowledge discovery and data mining*, pages 994–1002. ACM, 2012. 1.2

[69] Arka A Bhattacharya, Dezhi Hong, David Culler, Jorge Ortiz, Kamin Whitehouse, and Eugene Wu. Automated metadata construction to support portable building applications. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 3–12. ACM, 2015. 5.2.1

[70] Abdul Samad bin Haji Ismail, Abdul Hanan Abdullah, Kamalrulnizam bin Abu Bak, Md Asri bin Ngadi, Dahliyusmanto Dahlan, and Witcha Chimphlee. A novel method for unsupervised anomaly detection using unlabelled data. In *Proceedings of the International Conference on Computational Sciences and Its Applications*, pages 252–260. IEEE, 2008. 4.6

[71] Christoffer A Björkskog, Giulio Jacucci, Luciano Gamberini, Tatu Nieminen, Topi Mikkola, Carin Torstensson, and Massimo Bertoncini. Energylife: pervasive energy awareness for households. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing-Adjunct*, pages 361–362. ACM, 2010. 1.1

[72] Robert S Brewer and Philip M Johnson. Wattdepot: An open source software ecosystem for enterprise-scale energy data collection, storage, analysis, and visualization. In *Proceedings of the 1st IEEE International Conference on Smart Grid Communications (SmartGridComm),*, pages 91–95. IEEE, 2010. 2.6.2

[73] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009. 4.6

[74] Chao Chen and Diane J Cook. Energy outlier detection in smart environments. *Artificial Intelligence and Smarter Living*, 11:07, 2011. 4.6

[75] Chao Chen, Diane J Cook, and Aaron S Crandall. The user side of sustainability: Modeling behavior and energy usage in the home. *Pervasive and Mobile Computing*, 2012. 3.1.1, 3.2.2

[76] Dong Chen, Sean Barker, Adarsh Subbaswamy, David Irwin, and Prashant Shenoy. Non-intrusive occupancy monitoring using smart meters. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, pages 1–8. ACM, 2013. 3.1.2

[77] Jie Chen and Arjun K Gupta. *Parametric statistical change point analysis: with applications to genetics, medicine, and finance*. Springer, 2011. 4.3.1

[78] Haibin Cheng, Pang-Ning Tan, Christopher Potter, and Steven Klooster. A robust graph-based algorithm for detection and characterization of anomalies in noisy multivariate time series. In *Proceedings of the IEEE International Conference on Data Mining Workshops*, pages 349–358. IEEE, 2008. 4.6

[79] Haksoo Choi, Supriyo Chakraborty, Zainul Charbiwala, and Mani Srivastava. Sensorsafe: a framework for privacy-preserving management of personal sensory information. *Secure Data Management*, pages 85–100, 2011. 2.2.2, 2.6.3

[80] Xingchen Chu, B Durnota, Rajkumar Buyya, et al. Open sensor web architecture: Core services. In *Proceedings of the fourth International Conference on Intelligent Sensing and Information Processing*, pages 98–103. IEEE, 2006. 2.6.2

[81] S. Darby. The effectiveness of feedback on energy consumption. *A Review for DEFRA of the Literature on Metering, Billing and direct Displays*, 486, 2006. 1.1, 3.1.2

[82] Stephen Dawson-Haggerty, Xiaofan Jiang, Gilman Tolle, Jorge Ortiz, and David Culler. smap: a simple measurement and actuation profile for physical information. In *Proc. of SenSys*, pages 197–210. ACM, 2010. 2.4.1, 3.3.1, 3.4.1

[83] Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler. Boss: building operating system services. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 443–457, 2013. 2.6.2, 3.1.1, 3.1.3, 3.6

[84] Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. An operating system for the home. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 25–25. USENIX Association, 2012. 2.6.2, 3.1.1, 3.6

[85] V.L. Erickson and A.E. Cerpa. Occupancy based demand response hvac control strategy. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, pages 7–12. ACM, 2010. 1.1, 3.1.2

[86] V.L. Erickson, MA Carreira-Perpinan, and A.E. Cerpa. Observe: Occupancy-based system for efficient reduction of hvac energy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks*, pages 258–269. IEEE, 2011. 1.1, 1.2, 3.1.1, 3.1.2

[87] M. Evans, B. Shui, and S. Somasundaram. Country report on building energy codes in india. Technical report, Pacific Northwest National Laboratory (PNNL), Richland, WA (US), 2009. 1.1

[88] Romain Fontugne, Jorge Ortiz, Nicolas Tremblay, Pierre Borgnat, Patrick Flandrin, Kensuke Fukuda, David Culler, and Hiroshi Esaki. Strip, bind, and search: a method for identifying abnormal energy consumption in buildings. In *Proceedings of the 12th international conference on Information processing in sensor networks*, pages 129–140. ACM, 2013. 1.1, 1.2, 1, 4.1, 4.5

[89] GeSI. *SMART 2020: Enabling the low carbon economy in the information age*. Climate Group, 2008. 1.1

[90] William I Grosky, Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. Senseweb: An infrastructure for shared sensing. *Multimedia, IEEE*, 14(4):8–13, 2007. 2.6.2

[91] Manish Gupta, Abhishek B Sharma, Haifeng Chen, and Guofei Jiang. Context-aware time series anomaly detection for complex systems. In *SDM 13 Workshop on Data Mining for Service and Maintenance*, page 14, 2013. 4.6

[92] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. 3.6

[93] George William Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891, 1992. 1.2, 3.1.2

[94] Halldór Janetzko, Florian Stoffel, Sebastian Mittelstädt, and Daniel A Keim. Anomaly detection for visual analytics of power consumption data. *Computers & Graphics*, 38: 27–37, 2014. 4.5

[95] Xiaofan Jiang. A High-Fidelity Energy Monitoring and Feedback Architecture for Reducing Electrical Consumption in Buildings. *PhD dissertation*, UC Berkeley 2010. 2.6.2

[96] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009. 8

[97] Wilhelm Kleiminger, Christian Beckel, Thorsten Staake, and Silvia Santini. Occupancy detection from electricity consumption data. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, pages 1–8. ACM, 2013. 3.1.2, 3.2.2

[98] Andrew Krioukov, Gabe Fierro, Nikita Kitaev, and David Culler. Building application stack (bas). In *Proc. of BuildSys*, pages 72–79. ACM, 2012. 2.1, 3.1.1, 3.6

[99] Youngki Lee, SS Iyengar, Chulhong Min, Younghyun Ju, Seungwoo Kang, Taiwoo Park, Jinwon Lee, Yunseok Rhee, and Junehwa Song. Mobicon: a mobile context-monitoring platform. *Communications of the ACM*, 55(3):54–65, 2012. 3.6

[100] J. Lu, D. Birru, and K. Whitehouse. Using simple light sensors to achieve smart daylight harvesting. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, pages 73–78. ACM, 2010. 1.1

[101] 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*, pages 211–224. ACM, 2010. 1.2, 3.1.1, 3.1.2, 3.2.2, 3.6

[102] Emiliano Miluzzo, Cory T Cornelius, Ashwin Ramaswamy, Tanzeem Choudhury, Zhigang Liu, and Andrew 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*, pages 5–20. ACM, 2010. 3.6

[103] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007. 8

[104] Balakrishnan Narayanaswamy, Bharathan Balaji, Rajesh Gupta, and Yuvraj Agarwal. Data driven investigation of faults in hvac systems with model, cluster and compare (mcc). In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pages 50–59. ACM, 2014. 1.1

[105] R. Newton, L. Girod, M. Craig, S. Madden, and G. Morrisett. WaveScript: A Case-Study in Applying a Distributed Stream-Processing Language. *system*, 1(2008/1):31, 2008. 2.2.2

[106] Shahriar Nirjon, Robert F Dickerson, Philip Asare, Qiang Li, Dezhi Hong, John A Stankovic, Pan Hu, Guobin Shen, and Xiaofan 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*, pages 403–416. ACM, 2013. 3.6

[107] Los Angeles Department of Water and Power. New watering schedule. `http://www.ladwpnews.com/go/doc/1475/881355/New-Watering-Schedule-Now-in-Effect-for-LADWP-Customers`. [Online; accessed July 2013]. 3.1.2

[108] James Pierce and Eric Paulos. Beyond energy monitors: interaction, energy, and emerging energy systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 665–674. ACM, 2012. 1.1

[109] Huida Qiu, Yan Liu, Niranjan A Subrahmanya, and Weichang Li. Granger causality for time-series anomaly detection. In *Proceedings of the 12th International Conference on Data Mining*, pages 1074–1079. IEEE Computer Society, 2012. 4.6

[110] IEE Report. Utility-scale smart meter deployments: A foundation for expanded grid benefits. `http://www.edisonfoundation.net/iee/Documents/IEE_SmartMeterUpdate_0813.pdf`, 2013. [Online; accessed January 2015]. 4.1

[111] Anthony Rowe, Mario E Berges, Gaurav Bhatia, Ethan Goldman, Raj Rajkumar, James H Garrett, Jos MF Moura, and Lucio Soibelman. Sensor andrew: Large-scale campus-wide sensing and actuation. *IBM Journal of Research and Development*, 55(1.2):6–1, 2011. 2.6.2

[112] A. Schoofs, D.T. Delaney, G.M.P. O'Hare, and A.G. Ruzzelli. COPOLAN: non-invasive occupancy profiling for preliminary assessment of HVAC fixed timing strategies. In *Proceedings of the 3rd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*. ACM, 2011. 1.1, 3.1.2

[113] James Scott, AJ Bernheim Brush, John Krumm, Brian Meyers, Michael Hazas, Stephen

Hodges, and Nicolas Villar. Preheat: controlling home heating using occupancy prediction. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 281–290. ACM, 2011. 3.1.2

[114] John E Seem. Using intelligent data analysis to detect abnormal energy consumption in buildings. *Energy and Buildings*, 39(1):52–58, 2007. 1.1, 1.2, 1, 4.1, 4.6

[115] Rayman Preet Singh, Srinivasan Keshav, and Tim Brecht. A cloud-based consumer-centric architecture for energy data analytics. In *Proceedings of the fourth international conference on Future energy systems*, pages 63–74. ACM, 2013. 2.6.2, 3.1.1

[116] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Statistical change detection for multi-dimensional data. In *Proceedings of the 13th international conference on Knowledge discovery and data mining*, pages 667–676. ACM, 2007. 4.3.1

[117] L.V. Thanayankizil, S.K. Ghai, D. Chakraborty, and D.P. Seetharam. Softgreen: Towards energy management of green office buildings with soft sensors. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, pages 1–6. IEEE, 2012. 3.1.2

[118] Thomas Weng, Bharathan Balaji, Seemanta Dutta, Rajesh Gupta, and Yuvraj 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*, pages 13–18. ACM, 2011. 1.2

[119] Thomas Weng, Anthony Nwokafor, and Yuvraj 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*, pages 1–8. ACM, 2013. 3.1.1, 3.6

[120] Tri Kurniawan Wijaya, Tanuja Ganu, Dipanjan Chakraborty, Karl Aberer, and Deva P Seetharam. Consumer segmentation and knowledge extraction from smart meter and survey data. In *Proceedings of the SIAM International Conference on Data Mining (SDM14)*. SIAM, 2014. 4.2