SensorAct: A Privacy and Security Aware Federated Middleware for Building Management

Pandarasamy Arjunan[†], Nipun Batra[†], Haksoo Choi[§], Amarjeet Singh[†], Pushpendra Singh[†], Mani B. Srivastava[§]

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

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

Abstract

The archaic centralized software systems, currently used to manage buildings, make it hard to incorporate advances in sensing technology and user-level applications, and present hurdles for experimental validation of open research in building information technology. Motivated by this, we — a transnational collaboration of researchers engaged in development and deployment of technologies for sustainable buildings — have developed *SensorAct*, an open-source federated middleware incorporating features targeting three specific requirements: (i) Accommodating a richer ecosystem of sensors, actuators, and higher level third-party applications (ii) Participatory engagement of stakeholders other than the facilities department, such as occupants, in setting policies for management of sensor data and control of electrical systems, without compromising on the overall privacy and safety, and (iii) Flexible interfacing and information exchange with systems external to a building, such as communication networks, transportation system, electrical grid, and other buildings, for better management, by exploiting the teleconnections that exist across them. SensorAct is designed to scale from small homes to network of buildings, making it suitable not only for production use but to also seed a global-scale network of building testbeds with appropriately constrained and policed access. This paper describes SensorAct's architecture, current implementation, and preliminary performance results.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design, Architecture, Deployment

Buildsys'12, November 6, 2012, Toronto, ON, Canada.

Keywords

Middleware, Building management, Sensors, Actuators

1 Introduction

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. Computation technologies promise enabling intelligent sensing, learning, prediction, control, and actuation to be deeply embedded in the fabric of building. Such close integration will result in meeting performance goals with the smallest energy footprint while being responsive to changing usage pattern, external conditions and occupant needs. As a result there is now tremendous research and entrepreneurial activity, as exemplified by the *ACM BuildSys* itself, targeting buildings as an exciting substrate for novel computational methods and technologies that measure, model, understand, and optimize the design and operation of these complex human-cyber-physical systems.

However, these efforts in building-scale energy management are fraught with deployment challenges, many of which are rooted in the inflexible, isolated, slow, and archaic software that forms the *middleware* in the current building management systems. Designed for centralized facilities management departments, currently available middleware are poorly suited to accommodate the needs of emerging research and production systems. Emerging needs include (i) accommodating a richer ecosystem of hard and soft sensors, actuators, and higher level third-party applications, thus helping unleash innovation similar to the one observed in mobile phone segment (ii) participatory engagement of stakeholders other than the facilities department, such as occupants, in setting policies for control of electrical systems and sharing and management of sensor data, without compromising on the overall operational security and integrity, and (iii) flexible interfacing and information exchange with systems external to a building, such as communication networks, transportation system, electrical grid, water network, and other buildings for better management by exploiting the teleconnections that exist across them.

In order to nurture future information technologies for building management and create a robust research ecosystem, it is also important that middleware also provide mechanisms for experimental and research systems to interface

with a building for sensor information gathering and subsystem control in a suitably limited and controlled fashion taking into account safety, security and privacy considerations of the occupants. Such an ability of the middleware to be research-friendly is important since the complex physical envelope and social milieu that buildings are embedded in, makes them hard to model or to replicate in isolated testbeds, and controlled evaluation of new technologies in operational buildings is essential to scale and transition them into practice. The preceding observations are informed by our own first hand experiences in research involving - development of new sensing and actuation technologies relating to energy and water sustainability; their deployment in diverse testbed (single-family home, student apartments, student dormitories, offices, and dry laboratories) across two cities in two different countries with quite different climatic, economic, legal, social, and cultural settings; and, sharing of testbed access and data across the two different institutions involved.

This paper presents *SensorAct*, a middleware for building management that incorporates several interesting capabilities to help meet the requirements discussed above. SensorAct has a twofold vision. First, it seeks to be a platform for production use, providing the thin waist of an hourglass stack for building information technology systems. It enables a diversity of sensors/actuators and applications to sit below and above it respectively, and a participatory approach to building management that engages various stakeholders with overlapping concerns. Second, by enabling a web of interconnected installations, it seeks to seed a global-scale network of building testbeds with suitably constrained and policed access (to meet privacy and safety concerns of occupants, facilities management, and Institutional Review Board requirements for human subjects research) to sensor and actuator resources enabling validation, comparison, and generalization of research. The key features of SensorAct, described in depth later in the paper, are:

A tiered and distributed architecture consisting of *Virtual Personal Device Servers* for local storage, access control and device management, and a federation of *Brokers* as a distributed registry of users and device servers resulting in an architecture that combines global access with local control. Local device servers not only permit scaling to higher rate sensing and actuation, but also make it easier to meet safety and privacy concerns.

Powerful sensor and actuator guard rules that enable a firewall using which device owners and occupants of a space may exercise fine-grained control over sensory data management and control of building subsystems.

Lightweight tasking framework that enable end-applications to inject one-shot and persistent event-triggered scripts into the middleware to perform rich forms of sensor querying, sensor processing and fusion, actuator control, and notifications.

2 Related Work

Sensing and control in buildings is addressed at different levels by several systems, that can be broadly classified into four categories - Commercial Building Management Systems (BMS), Home Automation Systems, third party Data Aggregation Services, and Academic Research Systems. We briefly describe each one of these together with related work in privacy-enabling systems for sensor data collections.

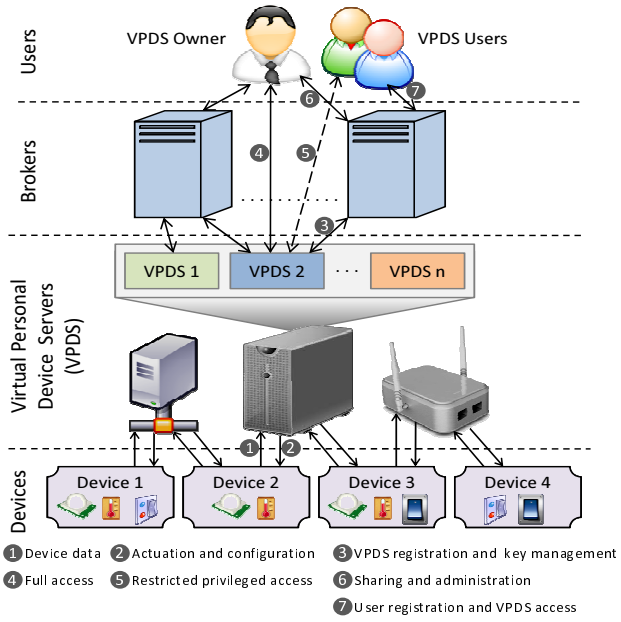


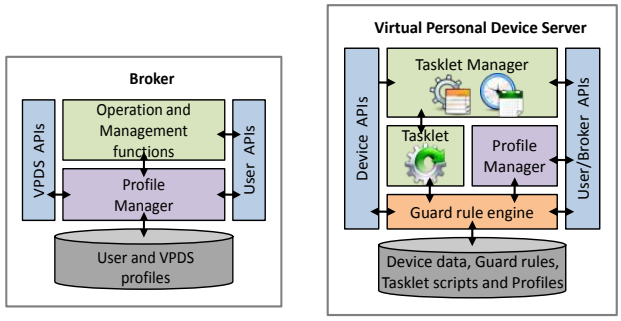
Figure 1: Tiered architecture of SensorAct System

Commercial BMS: have emerged as an end-to-end system for management of several building operations e.g. elevator control, fire response, and Heating Ventilation and Air Conditioning (HVAC) control. Of particular reference to our work is HVAC control. Several aspects differentiate the SensorAct system from BMS[1] enabled HVAC systems. Software architecture of these systems is targeted towards a centralized control with limited conflict resolution when multiple entities execute control over the same hardware controllers, thus restricting engagement of end user occupant. Further, these software systems expose limited hardware controller capabilities, allowing for control operations that are hard coded in the closed, vendor provided software system. While it is technically feasible for external devices and third-party applications to exchange sense and control information with the BMS systems using open protocols such as BACNet and OPC (Openness, Productivity and Collaboration) standard tunnels, complexity of these protocols and limited exposure of hardware controller capabilities restrict the support for multitude of new sensors (e.g. communicating over ZigBee and Z-Wave), actuators and third-party applications. Their expensive cost structure and stovepipe system architecture make BMS inaccessible to individual homeowners and constrained for data and control sharing across multiple organizations.

Home Automation Systems: A separate segment of industry has catered to the requirement of individual homeowners providing integrated hardware and software systems[2]

[1]Examples of commercially available BMS include http://www.trane.com, http://www.johnsoncontrols.com, http://www.buildingtechnologies.siemens.com

[2]http://micasaverde.com, http://www.homeseer.com, http://



(a) Broker Components (b) VPDS Components

Figure 2: Components of Broker and VPDS

and systems primarily providing software control with flexibility to integrate with diverse hardware systems[3]. These systems are primarily targeted towards security of the homes and comfort of its occupants rather than energy conservation. Visualization and script support in many of these systems provide users with increased engagement. While many of these systems support local data storage, there is limited support for sharing sense and control across multiple home deployments of these systems.

Data Aggregation Services: Several web-services have been introduced recently that allow data aggregation and visualization[4]. However, these services only provide limited support for actuation, primarily catering to messaging triggers. While RESTful interface allow these services to aggregate data from geographically separated sensors, centralized data storage architecture results in these services imposing significant sensor data rate limits. Even though their centralized architectures allow easy sharing of data across multiple users, these systems provide limited or no support for privacy and security of the end user thus restricting the data sharing to "all-or-nothing" model, rather than controlling the information released as function of context and sensor value.

Research systems: Several sensor based research systems, such as SenseWeb [11], SensorWeb [5] and GSN [1], have been developed and deployed. However, these systems are limited primarily to data aggregation, visualization and minimal sharing capabilities. There is recent research work [6, 7, 10] in development of generic systems for building automation. These systems lack comprehensive support for privacy aware data and control sharing across multiple users. Further, complex sense-and-control operations are only supported through applications external to the system, thus limiting the control loop bandwidth. In addition to overcoming these shortcomings, SensorAct system can also be easily accommodate other capabilities supported by these systems, such as metadata specification [6] and protocol adaptations and abstractions to interface with diverse sensor types [7]. Sensor Andrew [10] has similar goals as SensorAct but does not support fine-grained sharing and access control and limited database logging capability for high-frequency sensors. IBM's InfoSphere[5] platform provides a set of commercial software tools for data integration, information management and real time stream data analytics. But these tools were designed for enterprises and provides no support for data and control sharing capabilities.

Privacy-aware mobile data collection: Primary inspiration from the literature in privacy is that the users should be given appropriate control over sharing of their data and actuation of their devices [12]. This philosophy has been earlier adopted in other storage systems, in mobile data collection [3, 4, 6]. These systems propose personal data repositories of sensory data, using virtualization as a way to eliminate exposing data to third parties as would be the case with data aggregation services. Building on this inspiration, SensorAct system separates "hosting" from "sharing", unlike data aggregators wherein a central data storage does not allow fine-grained control over release of information or control of actuators. Privacy notions in SensorAct system architecture differ from these earlier proposed systems to additionally allow for security of actuators and data transformation mechanisms before releasing the data.

3 SensorAct Architecture

SensorAct is based upon tiered architecture as shown in Figure 1. The main components of SensorAct are Virtual Personal Device Server (VPDS) and Broker, interacting with devices at the lower layer and users and third-party applications at the higher layer.

3.1 Devices

A *device* can consists of hard and soft sensors and actuators. Sensory interface allows users to read the current state e.g. PIR sensor specifying motion. Actuators allow users to change the state of an appliance e.g. switching on the air-conditioner as well as setting the parameters for a sensory interface e.g. setting the sampling rate of a sensor. A device consisting of an actuator should specify an IP address for its accessibility. Each device is owned by a single user. Besides, the physical sensors and actuators, SensorAct also supports:

Computed sensors: are abstractions that can be calculated based on a pre-defined mathematical function applied on 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 motion sensor and microphone sensor).

Grouped actuators: are abstractions that can control multiple actuator 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.

3.2 Virtual Personal Device Servers (VPDS)

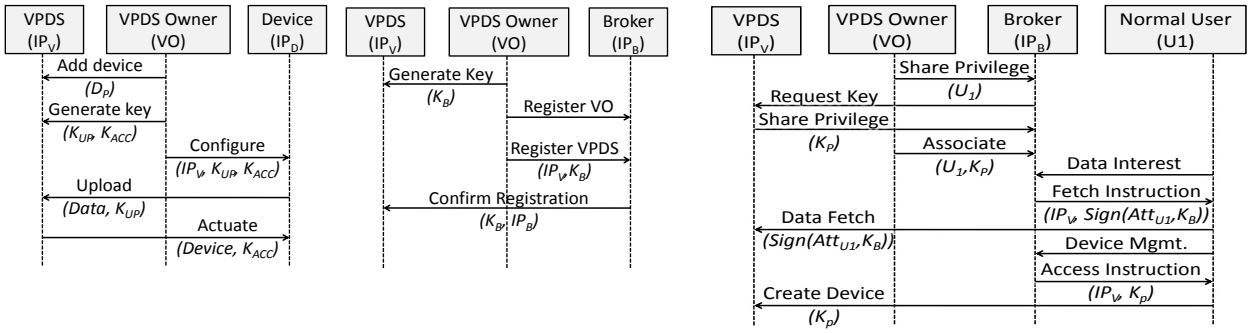
VPDS consists of a database, a *Tasklet manager*, a *Guard rule engine*, a profile manager and APIs for devices, users, and brokers to interact with the VPDS, as shown in Figure 2b. A single physical server may host multiple

www.control4.com/residential

[3] http://www.perceptiveautomation.com,http://www.eragy.com

[4] http://www.eragy.com/MyEragy/, https://cosm.com, http://www.nimbits.com, https://www.thingspeak.com, https://code.google.com/p/wattdepot/ [2]

[5] http://www.ibm.com/software/data/infosphere



(a) Device registration (b) VPDS registration (c) Sharing Privileges

Figure 3: Message exchange between VPDS Owner, VPDS, Broker and other users for different operations in the SensorAct system

VPDS, however virtualization is used to ensure the isolation of VPDS and privacy of the data within it. A single user owns a particular VPDS; though, the owner may allow other users to have controlled access to the sensors and actuators associated with the VPDS.

A VPDS Owner (VO) can manage multiple devices owned by him through the *profile manager*. A device profile consists of a set of attributes such as its name, IP address, a collection of sensor and/or actuator profiles, number of channels, data types and units, location, placement, and exposure. Additionally, a set of tags as $< key, value >$ pairs may be associated to let the user flexibly annotate additional details of the device. Device profile is represented using hierarchical model to name and identify devices, sensors, actuators, channels, and their readings uniquely e.g. $building : floor : room : device : id : [sensor|actuator] : id : channel : data$. A device search or data query can consists of a set of these attributes. VO also generates an upload key and an access key at the VPDS to be used in the devices for authorizing the data upload into the VPDS and actuation access by the VPDS respectively. Both the keys, together with the VPDS address are then configured manually in the device to register the device with the VPDS, as shown in Figure 3a. A device can also be associated with more than one VPDS. Once a device is registered with a VPDS, it can upload the data or take actuation commands from the corresponding VPDS.

A schema-less, key-value based database is used to store the sensor data from devices, allowing for efficient storage and querying of unstructured time-series data. The database stores WaveSegs (see Section 3.6) and can only be accessed via the *guard rule engine* in the VPDS. The guard rule engine is a set of guard rules (see Section 3.5) which are used by VO to restrict access to the sensory data and actuation, while sharing it with other users. *Tasklets* (see Section 3.7) are lightweight scripts used to perform rich forms of one-shot and persistent event-triggered actions including sensor querying, sensor processing and fusion, actuator control, and notifications. The triggers are in terms of complex events specified over the real-time clock and real-time sensor measurement. A task scheduler is used to execute the tasks.

The VPDS abstraction not only permits flexible provisioning of hardware resources but also allows diverse deployment scenarios ranging from VPDS hosted on a local machine for high-rate low-latency sensing and control to VPDS running on cloud-based hosting services such as Amazon EC with lower-rate sensing and control.

3.3 Broker

In SensorAct, a trusted broker contains a registry of users, a registry of VPDSs, and acts as a mediator to help clients establish a connection with VPDSs. Figure 3b illustrates the process for registering a VPDS to a broker. First, VO generates a key (K_B) on the VPDS. Secret key (K_B) together with VPDS attributes (e.g. IP address) are then used to register it with a broker. Upon registration, the broker contacts the VPDS with K_B and correspondingly the broker is added in the list of trusted brokers at the VPDS. A broker also allows a VO to share administrative privileges and data and control access, for his/her VPDS, with other users on the broker. A trusted broker is authorized to perform key management for the VO to share administrative privileges with other users. Broker also signs user attributes with a secret key for data or actuation access to the VPDS that is directly performed between the user and the VPDS. More details on privilege sharing is discussed in Section 3.5. In the future, we also plan to support federation of multiple brokers.

3.4 Users

In SensorAct, a user may have two roles: i) Owner of his/her own devices and correspondingly associated VPDS ii) User with data and control access as per the privileges assigned by the owner other devices. A single user could be the owner of his/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 the user devices or to even control them in a carefully constrained fashion. Users may use diverse end-applications to consume sensor data and control actuators in a device. These applications may provide users with more convenient interface to the underlying APIs and tasklet mechanism. SensorAct comes with a browser-based cloud-hosted application to let users interact with brokers and VPDS, but alternative end-applications can easily be implemented in diverse programming languages.

3.5 Sharing privileges

Two sets of privileges are associated with each VPDS - administrative privilege for guard rule, task and device management; and data and control access. When a VO wants to share some administrative privileges, the trusted broker

(where the VPDS is registered) contacts the corresponding VPDS and is provided with a secret key (K_P), as shown in Figure 3c. Each key provided by the VPDS has associated administrative privileges to perform one or more of the guard rule, task and device management operations. A single key may be shared across multiple users to provide them with same group privileges. When a user wants to perform administrative action, s/he requests for the VPDS address and privilege key from the broker and thereafter directly contacts the VPDS to perform the required action. Corresponding API request by the user at VPDS is authenticated using K_P.

API requests for data and control access are authenticated using user credentials, signed by the broker, and verified by a set of *"guard rules"*. Each VPDS internally maintains a set of guard rules, created by its owner and by other users to whom the owner has given the privilege to create guard rules. These guard rules are intended to meet privacy and safety concerns associated with sharing of one's sensors and actuators. For example, the expressive guard rules permit policies such as restricting access to a sensor by a user or group of users as a function of time of day, sensor location, and current measured value. This is considerably more powerful than the all-or-nothing access to a sensors data that current user authentication based data aggregation services and other research systems provide. Moreover, the guard rules also police access to the actuators, and ensure that a user may not issues unsafe commands or sequence of commands to an actuator. For example, facilities management may limit the range within which a user may set an actuator, or how frequently a user may change actuator settings.

By default, owner has all the privileges while other users have no access unless permitted by the guard rules. Hierarchy of users exercise control over shared devices using nested rules wherein conflicts are avoided using priorities associated with each rule. For the convenience of the VPDS owner, SensorAct system provides a set of template rules that can be filled in with required parameters to create concrete rules. Users can create their own template rules or use the template rules provided by the system. Further flexibility in guard rule definition is provided through support for globally defined macros and in-built functions for data obfuscation. Users can define their own groups of users using macros or specify multiple users by using regular expressions in user condition.

Figure 4 illustrates sample guard rules and associated architectural features. Figure 4a shows a guard rule that allows all users in SensorAct.edu domain to access data collected during work time only. However, the rule obfuscates the data by adding Gaussian noise (provided as an internal function by SensorAct system) with mean 70 and variance 15. *WORK_TIME* in condition field is a macro defined through corresponding API with JSON object shown in Figure 4b. To define repeat time, Unix Cron time expression is used to specify 9am to 6pm during weekdays. Figure 4c is an example of a template rule. The rule allows a user to change an actuators value within a certain range, which is parametrized. The concrete instantiation of the template rule is shown in Figure 4d, which specifies the parameter values as 60 and 100.

```
{
 "NAME": "AddNoiseRule",
 "TARGET_OPERATION": "READ",
 "DESCRIPTION": "Add gaussian noise to data on
                 work time for all users at SensorAct.edu."
 "PRIORITY": 1,
 "CONDITION": "USER.email == *@SensorAct.edu
               && TIME == WORK_TIME",
 "ACTION": "AddGaussianNoise(70, 15)"
}
```

(a) Normal guard rule

```
{ "MACRO_NAME": "WORK_TIME",
  "MACRO_VALUE": "[ * 9-18 * * 1-5 ]" }
```

(b) Guard rule macro

```
{
  "NAME": "ValueRangeTemplate",
  "DESCRIPTION": "Allow changing actuator
                  values within specified range."
  "TARGET_OPERATION": "WRITE",
  "CONDITION": "VALUE >= #PARAM_MIN_VALUE
                && VALUE <= #PARAM_MAX_VALUE",
  "ACTION": "ALLOW"
}
```

(c) Template guard rule

```
{
 "NAME": "ValueRangeRule",
 "TEMPLATE_RULE_NAME": "ValueRangeTemplate",
 "DESCRIPTION": "Allow changing actuator values
                 within 60 to 100."
 "PRIORITY": 1,
 "PARAMETERS": { "#PARAM_MIN_VALUE": 60
                 "#PARAM_MAX_VALUE": 100 }
}
```

(d) Concrete rule instantiated from the template rule

Figure 4: Example guard rules

3.6 Sensory Information Representation

An important aspect of SensorAct is that it 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 [4] and in turn inspired by *SigSeg* used in MIT's WaveScript system [9]. 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 5 illustrates the

```
{
  "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 5: JSON representation of a sample WaveSeg

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 affects query processing performance. To further optimize performance, SensorAct opportunistically merges WaveSegs as they are uploaded by a sensor.

3.7 Lightweight Tasking Framework

The tasking framework of SensorAct supports *tasklet*, light-weight non-blocking scripts, to perform sense-and-control operations within the system. Tasklets provide a generic way to perform several system operations such as querying current and historical sensor data, real time actuation, complex event management and processing, and customized notifications and alerts. The tasklet framework follows a simple programming model using the following primitives:

- *Input* - List of system devices and timers that act as triggers for the corresponding action specified in the tasklet. This is inspired by sensitivity list used in VHDL programming language.
- *When* - Specifies a complex boolean operation using AND, OR primitives as a combination of triggers specified as inputs.
- *Execute* - Specifies the corresponding action to be performed when the condition in the "When" primitive holds true.

Figure 6a shows an example tasklet. "Input" consists of a time based trigger that becomes active every 2 minutes between 10AM and 6PM everyday. "When" primitive specifies a simple condition which holds true every time the trigger is active. "Execute" primitive specifies a script *monitor_ac.lua* that is executed whenever "When" primitive is true, in this case every 2 minutes between 10 AM and 6 PM everyday. Additionally, parameter list using *PARAMS* primitive can be used for parameterized action scripts. Using various inputs, proposed tasklet model can support diverse operations as follows:

One-shot operations e.g. querying current status of a device are supported by specifying null triggering condition resulting in the action script getting executed immediately and only once.

Periodic operations are supported by specifying periodic timing conditions as triggers, as is also illustrated in the example discussed in Figure 6a.


```
{
  "NAME": "Monitor_AC",
  "PARAMS": {
    "T1": "Mickey:Room1:Tempr:1",
    "A1": "Mickey:Room1:AC:1",
    "MINS": 5,
    "LIMIT": 30
  }
  "INPUT": {
    "TIMER1": "[0 0/2 10-18 * *]"
  }
  "WHEN": "TIMER1",
  "EXECUTE": "[monitor_ac.lua]"
}
```


(a) Example tasklet format

```
-- monitor_ac.lua
-- Reads sensor readings and
-- actuate appliance accordingly

-- epoc MINS minutes before
epocNmin = os.time() - (60*MINS)

-- read MINS minutes avg value
avgTr = VPDS:readAvg(T1,epocNmin)

-- Check and turn ON
if avgTr > LIMIT then
    VPDS:write(A1,VPDS:TURNON)
end
```

(b) Tasklet scipt (Lua)

Figure 6: Tasklet example: Every 2 minutes from 10AM to 6PM every day, read the past 5 minutes average temperature of room1 and turn on the Air-Conditioner in the same room if the average temperature is above 30

Event driven operations e.g. real time monitoring and interactions with devices, are supported using only devices as triggers, resulting in the action script getting executed whenever a change in the device value is observed.

Periodic and Event based operations include fine-grained monitoring and control, are supported using a combination of devices and timers as triggers. In this case, the action script will be executed whenever a change in the state of the device is observed and/or the timing condition is satisfied (depending upon the boolean expression in the "When" primitive.

Action scripts are independent light-weight and non-blocking programs, very similar to interrupt service routines, that performs read and write operations on devices and computes simple calculations and branching using the read data. Any read/write operation to a device made by the tasklet pass through the guard rule engine, explained in Section 3.5. As a result, only those read/write operations that are authorized by the device owners will be allowed. The tasklets will inherit the permissions of the user who is executing it.

A *Tasklet Manager* receives the tasklet execution requests and executes a tasklet when its corresponding triggering conditions are met. Each tasklet runs with its own independent execution context which is monitored by the tasklet manager. As a result, no data exchange is supported across tasklets. Once a tasklet is submitted, the tasklet manager validates the task execution request and registers a trigger to the system that monitors for the events (timer and device data) based upon its triggering condition. A handle is assigned to the tasklet and is returned to the user for future reference that may include knowing the status or canceling the tasklet. Tasklet manager also keeps track of any data produced by the tasklet, particularly the data produces by periodic tasklets. This data is accessible by the user executing the tasklet using the handle of the tasklet.

4 Implementation

Implementation of the architecture, as proposed in Section 3, is an ongoing activity. We used Java and Play framework[6] to implement multiple components of VPDS and Broker and schema-less MongoDB as database. We released the system implementation code in open source[7] on web hosted code repository and welcome community contribution and

[6] http://www.playframework.org

[7] https://github.com/iiitd-ucla-pc3

Component	VPDS APIs
User	/user/{register|login|list}
Key	/key/{generate|delete|list|enable|disable}
Device	/device/{add|delete|get|list|search|share}
	/device/template/{add|delete|get|list}
Guardrule	/guardrule/{add|delete|get|list}
	/guardrule/association{add|delete|get|list}
Tasklet	/tasklet/{add|delete|get|list}
	/tasklet/{execute|cancel|status}
Data	/data/{upload/wavesegment|query}
Component	**Broker APIs**
User	/user/{register|login}
VPDS	/vpds/{register|remove}

Table 1: Supported SensorAct APIs for different system components

Figure 7: Snapshot of sample user interface web application

usage. Table 1 lists the SensorAct APIs currently supported in each of the system component. We also created a user interface application, using several Web 2.0 technologies, for VPDS owner to interact with the VPDS. Figure 7 shows the snapshot of a sample user interface web application. The current implementation of guard rule supports allow and deny actions on read and write operations with basic conditions. Arbitrary boolean expression are supported for conditions *USER.email*, *LOCATION_TAG* (exact match), *TIME* (unix epoch time) and *VALUE*.

The *Tasklet manager* uses Quartz[8], an industrial-strength task scheduler that scales well to large number of tasklets besides interfacing well with the underlying Java-based implementation of SensorAct. The scheduler supports both persistent and one-time tasks. In the current implementation, a background job is initiated on receiving a tasklet to monitor the corresponding triggers. Whenever the registered trigger is raised, corresponding tasklet will be executed in a separate worker thread, managed by tasklet manager.

We use Lua[9], a lightweight scripting language, to express tasklet script. Lua interpreters are known for small memory footprint and fast execution, and Lua itself lends well to domain-specific extensions. We use Java Scripting API framework to execute lua scripts through jnlua[10], a Lua script engine that integrates Lua into Java and supports language bindings. Figure 6 illustrates a sample tasklet execution request and the corresponding Lua script, demonstrating the flexibility and compactness of the SensorAct's scripting framework. Users can use the currently supported interface to write their own tasklets. We also plan to provide support for templates and graphical user interface to create common tasklets and guard rules.

[8] http://quartz-scheduler.org

[9] http://www.lua.org

[10] http://code.google.com/p/jnlua

5 Evaluation

The complete evaluation of SensorAct requires extensive deployment of hardware systems and a detailed user study regarding the the usability and utility of the system features. As this first work primarily focus upon the system architecture, detailed evaluation is outside the scope of this paper. In this section, we explain about our ongoing deployment plan and the performance evaluation of SensorAct by emulating the proposed deployment scenario.

5.1 Deployment plan

Proposed SensorAct will be deployed in diverse environments by multiple collaborators, including two academic partners - IIIT Delhi and UCLA and an industrial research laboratory - IBM Research, India. Our largest deployment in IIIT Delhi campus involves sensing infrastructure across 400 rooms in student dorms with hardware support for up to 10 channels of sensory data per room, monitoring various energy, ambient, and occupancy variables at 1 Hz. We further plan to extend the sensing infrastructure to monitor faculty offices and interact with commercial Building Management System (put in place for HVAC Control). Sensing and control for all the infrastructure will be done using proposed SensorAct system. Another academic deployment at UCLA involves a 1200 sq. ft. laboratory space with 30 electrical channels each and 10 event driven sensors for motion, door, and light. Proposed system deployment for Softgreen [13] testbed at IBM Research India involves more than 75 sensing nodes collecting temperature, motion and various soft sensors to infer employee occupancy information.

For each of the ongoing deployments, a multitude of sensing platforms are being used for monitoring diverse set of parameters. These platforms include Z-Wave based sensors and plug computers, panel monitors, water flow sensors and Ethernet/Wi-Fi based microcontroller platforms. We were able to easily collect data from all these diverse sensors into the SensorAct system with minimal effort. Experience with these diverse deployment experience shows the general applicability of the SensorAct system architecture.

5.2 System Performance

We experimented the performance and scalability of SensorAct by emulating our largest deployment scenario. In this setup, 400 identical processes running in parallel upload sensor readings, each sending 100 sensor measurements (10 sensing channels at 1 Hz) every 10 seconds in the form of a WaveSegs. All the data upload processes were hosted on a laptop connected over Intranet with another laptop (2.3 Ghz Intel Core i7 processor, 8 GB RAM, 5400 RPM Hard Disk) hosting SensorAct. We disabled the Guard Rule engine to test the baseline performance. We conducted this data upload experiment for 40 minutes. We verified that all the WaveSegs were aggregated. The average and median CPU usage was 14.6% and 7% and average and median RAM usage was 3.7% and 3.2% respectively. Results clearly indicate the scalability of the SensorAct system for large scale data aggregation from low frequency sensors.

The overhead of guard rule engine is based upon the number of rules, target operation (read or write) and number and size of WaveSegs. Tasklet performance is based upon the complexity of the script, size of object binding between Lua and Java, and the number of read/write requests that goes through the guard rule engine. To emulate a common read operation by a normal user, we setup three guard rules - one checking for the user email (at the WaveSeg level) and the remaining two checking for the timestamps and the sensor values (to be done for each reading in the WaveSeg). We emulated the read operation, querying for 15 minutes of 1 Hz sensor data as a nil triggered tasklet that reads 90 WaveSegs, each containing 10 data values.

In the first experiment, we scheduled to execute the tasklet once in every 5 seconds for a total period of 30 minutes. We observed that, on an average, guard rule engine takes 3356ms (including 10ms for database read) to process guard rules for each tasklet and each tasklet execution instance takes 3363ms (including guard rule processing time). Since this operation will be infrequent, current processing time will work fine. However, we will be working on improving the guard rule processing as proposed in the future work. To emulate the data access by VPDS Owner, we repeated the experiment without the guard rules. In this case, on an average, guard rule engine takes 11ms to process guard rules for each tasklet and each tasklet execution instance takes 11.5ms (including guard processing time). Average CPU usage during these experiments were 68% and 2% respectively on a laptop (2.4GHz Intel Core i5 processor and 2GB RAM).

6 Conclusions and Future Directions

SensorAct system is a result of the needs that we faced in our own transnational collaborative research: an open, flexible, taskable, and scalable information substrate for buildings into which multitude of sensors, actuators, and applications could be integrated; a building management system approach in which the occupants are engaged as participatory stakeholders instead of just passive consumers; and, exchange of data and limited access to testbeds in a fashion that was cognizant of IRB constraints and sensitive to occupants privacy and buildings safety. These experiences motivated the proposed architectural design of SensorAct system.

While the initial implementation is done and in early stages of being incorporated into our deployments, SensorAct itself continues to evolve. Some specific areas of development include seamless and intuitive device management functions (automatic new device registration and removal of obsolete devices), location and context based device search and data query, implementation of federated broker aspect of the architecture, performance optimization of the guard rules via methods such as dynamic code compilation techniques in packet filters used for network firewalls [8, 14], and supplementing guard rules which target behavioral privacy with an anonymizing and aggregating proxy for privacy of identity. Moreover, with the development code tree open source and hosted on a cloud-based hosting and version control service, we hope to engage other researchers as contributors of code to SensorAct and to link their testbeds via SensorAct.

7 Acknowledgments

This work is partially supported through Indo-US PC3 collaborative program, supported by NSF, USA (Grant Number CNS-1143667) and DEITy, India. We also acknowledge support of IBM Research, India for using SensorAct for their data collection and for partial support of Pandarasamy Arjunan through IBM PhD Fellowship. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

8 References

[1] K. Aberer, M. Hauswirth, and A. Salehi. Global Sensor Networks. In *Technical report LSIR-REPORT-2006-001*, 2006.

[2] R. Brewer and P. Johnson. WattDepot: An Open Source Software Ecosystem for Enterprise-Scale Energy Data Collection, Storage, Analysis, and Visualization. In *First IEEE International Conference on Smart Grid Communications*, SmartGridComm, 2010.

[3] R. Cáceres, L. Cox, H. Lim, A. Shakimov, and A. Varshavsky. Virtual Individual Servers as Privacy-Preserving Proxies for Mobile Devices. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, MobiHeld, 2010.

[4] H. Choi, S. Chakraborty, Z. M. Charbiwala, and M. B. Srivastava. SensorSafe: a Framework for Privacy-Preserving Management of Personal Sensory Information. In *Proceedings of the 8th VLDB international conference on Secure data management*, SDM, 2011.

[5] X. Chu, T. Kobialka, B. Durnota, and R. Buyya. Open Sensor Web Architecture: Core Services). In *4th International Conference on Intelligent Sensing and Information Processing*, ICISIP, 2006.

[6] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. sMAP: a Simple Measurement and Actuation Profile for Physical Information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys, 2010.

[7] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and V. Bahl. An Operating System for the Home. NSDI, USENIX, 2012.

[8] D. Engler and M. Kaashoek. DPF: Fast, Flexible Message Demultiplexing using Dynamic Code Generation. In *ACM SIGCOMM Computer Communication Review*, volume 26, pages 53–59. ACM, 1996.

[9] 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.

[10] A. Rowe, M. Berges, G. Bhatia, E. Goldman, R. Rajkumar, J. H. Garrett, J. M. F. Moura, and L. Soibelman. Sensor Andrew: Large-scale Campus-wide Sensing and Actuation. *IBM Journal of Research and Development*, 2011.

[11] A. Santanche, S. Nath, J. Liu, B. Priyantha, and F. Zhao. SenseWeb: Browsing the Physical World in Real Time (Demo Abstract). In *ACM/IEEE Information Processing in Sensor Network*, IPSN 2006.

[12] K. Shilton, J. Burke, D. Estrin, R. Govindan, M. Hansen, J. Kang, and M. Mun. Designing the Personal Data Stream: Enabling Participatory Privacy in Mobile Personal Sensing. TPRC, 2009.

[13] L. V. Thanayankizil, S. K. Ghai, D. Chakraborty, and D. P. Seetharam. Softgreen : Towards Energy Management of Green Office Buildings with Soft Sensors. In *Fourth International Conference on Communication Systems and Networks*, COMSNETS, 2012.

[14] Z. Wu, M. Xie, and H. Wang. Swift: A Fast Dynamic Packet Filter. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 279–292. USENIX Association, 2008.