# An Interoperable Realization of Smart Cities with Plug and Play based Device Management

Prasant Misra[†], Vasanth Rajaraman[†], Kumaresh Dhotrad[†], Jay Warrior[†], Yogesh Simmhan[‡]

[†]Robert Bosch Centre for Cyber Physical Systems, Indian Institute of Science, Bangalore, India
[‡]Supercomputing Education and Research Centre, Indian Institute of Science, Bangalore, India
Email: {prasant.misra, vasanth.rajaraman, jay.warrior}@rbccps.org[†] simmhan@serc.iisc.in[‡]

*Abstract*—The primal problem with Internet of Things (IoT) solutions for smart cities is the lack of interoperability at various levels, and more predominately at the device level. While there exist multitude of platforms from multiple manufacturers, the existing ecosystem still remains highly closed. In this paper, we propose SNaaS or Sensor/Network as a Service: a service layer that enables the creation of the plug-n-play infrastructure, across platforms from multiple vendors, necessary for interoperability and successful deployment of large-scale city wide systems. In order to correctly position the new service layer, we present a high level reference IoT architecture for smart city implementations, and follow it up with the workflow details of SNaaS along with preliminary microbenchmarks.

*Keywords*—*IEEE* 1451, *Plug and Play, Self describing devices, IoT reference architecture, Smart cities*

## I. INTRODUCTION

The world is undergoing an unprecedented pace of urbanization, and if present trends continue, the world urban population will rise by about 72% in the next 4 decades. While the reasons for this rapid urban expansion are many, it is primary driven by two entities with complementary needs: *cities*, on one end, to attract the best skilled people and enterprises; and *people*, on the other end, by their preference to migrate to cities that provide better quality of life. This rapid scale of urbanization will need smarter, sustainable cities that are able to effectively and efficiently manage city utilities and services for its citizens.

Electric grids, water distribution systems, transportation systems, communication infrastructure, waste treatment plants, commercial buildings, hospitals, homes and education centers are existing *vital* facilities that shape the liveability standard of a city. In the future, newer infrastructures and services will also bring benefits and create opportunities of added value to its citizens. An efficient and effective management of these existing and possibly new city systems requires *integration*, an aspect needed for *transforming* a traditional city into a smarter form. However, value creation through integration can *only* be delivered with compatibility of technologies that is best achieved through standards that ensure *interoperability*.

The technology revolution driven by the Internet of Things, or the IoT offers promising smart city solutions. IoT, in its basic definition, is a vision for a ubiquitous society wherein people and "Things" are connected in an immersively networked computing environment. However, current IoT-based smart city implementations have mainly concentrated on vertical integration of independent infrastructure and services for siloed applications (Table I). Available solutions are extremely

*closed* with an ecosystem that is highly *locked-in* by vendors, wherein a single vendor owns the vertical application, platform, communication, services, and data. While interoperability will ensure better system management and open markets to competitive solutions that are affordable and sustainable, the existing ecosystem allows minimal/no flexibility.

In this paper, we explore the interoperability problem at the "Things" (i.e., device) level. "Things" or physical modules/platforms contain the interface to the physical world for measurement of various physical parameters. Due to interoperability constraints, the existing IoT ecosystem does not have provisions for interfacing devices of varying heterogeneity and complexity from multiple manufacturers that use different proprietary middleware technologies. In order to overcome this fundamental limitation, we propose SNaaS.

Expanded as Sensor/Network as a Service, SNaaS enables the creation of the plug-n-play infrastructure across platforms from multiple vendors, necessary for interoperability and successful deployment of large-scale city wide systems. SNaaS abstracts the features, capabilities and controls of the sensors and communication layer to allow higher level services to be developed and interacts with sensors/networks in a generalizable manner. In our instantiation of SNaaS, we leverage the IEEE 1451 [1] standard (originally developed for industrial processes automation) for describing the capabilities of the sensors/networks.

The rest of the paper unfolds as follows. To ground our discussion, we provide a high level reference IoT architecture to understand the rightful placeholder of SNaaS in the smart city framework in the next section. It is followed by Section III that presents the architecture and implementation details of SNaaS with preliminary microbenchmarks.

## II. A REFERENCE IoT ARCHITECTURE FOR SMART CITIES

In this section, we present one possible IoT architecture based on a 4-layered stack that reflects modular design principles for smart city implementations.

- **Layer-1: Physical/Virtual Space.** The lowest layer is a collection of sensing elements, or data generators and consumers that provide *context* information. This information is sensed not only from the physical space of hardware-level sensors (that are deployed as part of static infrastructure or mobile entities), but also from soft sensors that exist in the virtual space such as aggregated event streams and crowd-sourced information.

- **Layer-2: SNaaS.** The SNaaS layer consists of control and data planes. The data plane will offer channel models (e.g., event driven, sample and hold, etc.,) as a universal

TABLE I: **IoT based solutions for siloed smart city applications**

| | Smart Bus Stop | Smart Parking | Traffic Management | Smart Grid | Pollution Monitoring | Smart Street Lightning | Smart Garbage | Smart Healthcare | Smart Citizen |
|---|---|---|---|---|---|---|---|---|---|
| **Barcelona, Spain** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| **Southampton, UK** | | | | | | ✓ | | | ✓ |
| **Bristol, UK** | ✓ | | ✓ | | ✓ | | | | |
| **Glassgow, UK** | | | | | ✓ | ✓ | | ✓ | ✓ |
| **Chicago, US** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| **Sacramento, US** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | |

abstraction for input/output to the physical and virtual space of the sensing layer. The control plane will be responsible for managing the sensor/ network and providing a standardized life-cycle for discovery, configuration and use of the channel models. The combination is enabled by the creation of a TEDS-based plug-n-play infrastructure across platforms from multiple vendors. This information will either be broadcasted by the new sensing entity to the infrastructure elements, or actively pulled by the infrastructure itself from a TEDS repository on receipt of the sensor identifier. Providing a self-describing mechanism to characterize the transducers will allow the SNaaS manager to record its properties and semantic description. The control plane will expose ways to turn on/off sensors or specific attributes, change their sampling interval, etc.,.

Following the configuration of the "Thing", the data plane will publish the observations of each sensor as individual data streams to the upper layer. The combination of semantic capability and data push/pull (with a publish/subscribe model) will allow "Things" to be assembled into a semantically meaningful data flow graphs [2]. The SNaaS layer will, therefore, act as a registry for semantic discovery and linkage of "Things", and for virtualizing the real world.

● **Layer-3: "Big-Little" Data Management.** SNaaS offers a conduit for streaming "Little" data from distributed physical and virtual sensors. This data has to be cataloged, curated; and if necessary, persisted and aggregated, in order to perform subsequent analytics. The data management layer helps discover and maintain a *registry* of data sources and their characteristics such as periodicity, liveliness, quality, etc., and make them available for subsequent analysis. Access to data may be as streams of events pushed using a *publish-subscribe* model, pulled on-demand, or accumulated at the device. These may be controlled using *data privacy* and distributed access control policies that are enforced by this layer.

Data management should also help integrate with aggregate and slow-changing "Big" data that is available from canonical sources or have accumulated from sensor streams. "Big" data is hosted at central locations, globally or at local caches. Integrating "Big" with "Little" data requires a common vocabulary that helps to bridge the semantics. Data quality checks and validation are needed to understand the usability of data for specific needs.

A *data brokering service* that helps to interface data consumers with data producers is enabled using the building blocks of data ownership, incentives to collect data, data description, data quality and access control. The broker can help incentivize data collection and reuse through attribution, barter, or even monetary rewards.

● **Layer-4: Analytics and Decision Making.** Analytics for IoT applications often fall into categories of data and pattern mining, predictive analytics and forecasting, event and pattern detection, and optimizations. While there may be domain/application specific analytic, it is useful to have key analytics algorithms available that can be reconfigured to suit common needs. Time series and regression tree forecasting are useful for predicting future conditions. Pattern mining and clustering can help group conditions that exhibit similar behavior so that collective action can take place, or extrapolation from one entity in the cluster to the rest can happen (e.g., recommendations) [3]. Mining can also help identify causality between a pattern of features and an event of interest. Such patterns and predictions can help feed into real-time complex event pattern matching [4] that detect situations of interest and help respond to, or preferable, preempt them. These also feed into optimization algorithms that can control physical or virtual "Things" to ensure reliability and efficiency of the system.

We refer our readers to [5], [6] for additional details about this reference IoT architecture.

## III. The Design of SNaaS

In order to make this section self-contained, we first provide a brief overview of TEDS defined in the IEEE 1451 standard, and then present the overall architecture of SNaaS with preliminary microbenchmarks.

### A. Self describing devices with TEDS

TEDS is the electronic version of the data sheet that we use to configure a sensor. TEDS brings forward the concept that if the data sheet is electronic and can be readily accessed upon sensor discovery, it would be possible to configure the sensor automatically. This is analogous to the operation of plugging a mouse, keyboard, or monitor in the computer and using them without any kind of manual configuration. Thus, in principle, TEDS enables self-configuration of the system by self-identification and self-description of sensors and actuators (i.e., plug-and-play).

The IEEE 1451 [1] standard defines TEDS as a collection of multiple sections (such as sensor identification, calibration, correction data and manufacturer-related information) to form a generic electronic data sheet. IEEE 1451 divides the system into two general category of devices: (1) transducer interface module (TIM), which provide the necessary interfacing mechanism/functional units (such as the physical interface, signal conditioners, ADC/DAC convertors) for sensors and actuators; and (2) network capable application processor (NCAP), which
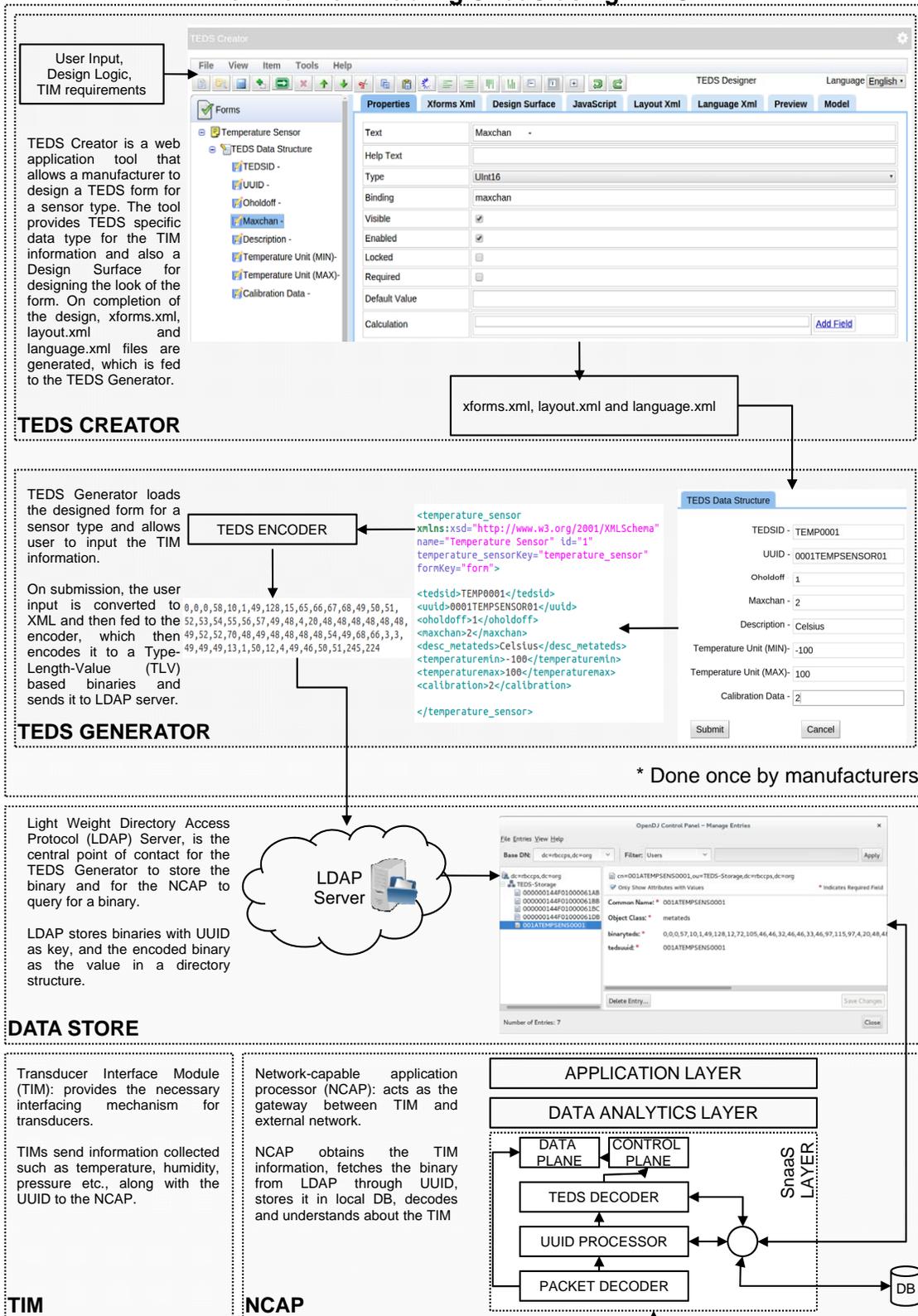
**Workflow for Enabling SNaaS using TEDS**

## TEDS CREATOR

User Input, Design Logic, TIM requirements

TEDS Creator is a web application tool that allows a manufacturer to design a TEDS form for a sensor type. The tool provides TEDS specific data type for the TIM information and also a Design Surface for designing the look of the form. On completion of the design, xforms.xml, layout.xml and language.xml files are generated, which is fed to the TEDS Generator.

TEDS Creator

File  View  Item  Tools  Help

TEDS Designer                Language  English ▾

Forms
- Temperature Sensor
  - TEDS Data Structure
    - TEDSID -
    - UUID -
    - Oholdoff -
    - Maxchan -
    - Description -
    - Temperature Unit (MIN)-
    - Temperature Unit (MAX)-
    - Calibration Data -

Properties | Xforms Xml | Design Surface | JavaScript | Layout Xml | Language Xml | Preview | Model

| | |
|---|---|
| Text | Maxchan    - |
| Help Text | |
| Type | UInt16 |
| Binding | maxchan |
| Visible | ☑ |
| Enabled | ☑ |
| Locked | ☐ |
| Required | ☐ |
| Default Value | |
| Calculation | Add Field |

xforms.xml, layout.xml and language.xml

## TEDS GENERATOR

TEDS Generator loads the designed form for a sensor type and allows user to input the TIM information.

On submission, the user input is converted to XML and then fed to the encoder, which then encodes it to a Type-Length-Value (TLV) based binaries and sends it to LDAP server.

TEDS ENCODER

```
<temperature_sensor
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
name="Temperature Sensor" id="1"
temperature_sensorKey="temperature_sensor"
formKey="form">

<tedsid>TEMP0001</tedsid>
<uuid>0001TEMPSENSOR01</uuid>
<oholdoff>1</oholdoff>
<maxchan>2</maxchan>
<desc_metateds>Celsius</desc_metateds>
<temperaturemin>-100</temperaturemin>
<temperaturemax>100</temperaturemax>
<calibration>2</calibration>

</temperature_sensor>
```

0,0,0,58,10,1,49,128,15,65,66,67,68,49,50,51,
52,53,54,55,56,57,49,48,4,20,48,48,48,48,48,48,
49,52,52,70,48,49,48,48,48,48,54,49,68,66,3,3,
49,49,49,49,13,1,50,12,4,49,46,50,51,245,224

TEDS Data Structure

| | |
|---|---|
| TEDSID - | TEMP0001 |
| UUID - | 0001TEMPSENSOR01 |
| Oholdoff - | 1 |
| Maxchan - | 2 |
| Description - | Celsius |
| Temperature Unit (MIN)- | -100 |
| Temperature Unit (MAX)- | 100 |
| Calibration Data - | 2 |

Submit          Cancel

\* Done once by manufacturers

## DATA STORE

Light Weight Directory Access Protocol (LDAP) Server, is the central point of contact for the TEDS Generator to store the binary and for the NCAP to query for a binary.

LDAP stores binaries with UUID as key, and the encoded binary as the value in a directory structure.

LDAP Server

OpenDJ Control Panel – Manage Entries

File  Entries  View  Help

Base DN:  dc=rbccps,dc=org     Filter: Users          Apply

- dc=rbccps,dc=org
  - TEDS-Storage
    - 000000144F01000061AB
    - 000000144F01000061BB
    - 000000144F01000061BC
    - 000000144F01000061DB
  - 001ATEMPSENS0001

cn=001ATEMPSENS0001,ou=TEDS-Storage,dc=rbccps,dc=org

☑ Only Show Attributes with Values                    \* Indicates Required Field

Common Name: \*   001ATEMPSENS0001
Object Class: \*   metateds
binaryteds: \*    0,0,0,57,10,1,49,128,12,72,105,46,46,32,46,46,33,46,97,115,97,4,20,48,48
tedsuuid: \*      001ATEMPSENS0001

Delete Entry...                                 Save Changes

Number of Entries: 7                              Close

## TIM

Transducer Interface Module (TIM): provides the necessary interfacing mechanism for transducers.

TIMs send information collected such as temperature, humidity, pressure etc., along with the UUID to the NCAP.

## NCAP

Network-capable application processor (NCAP): acts as the gateway between TIM and external network.

NCAP obtains the TIM information, fetches the binary from LDAP through UUID, stores it in local DB, decodes and understands about the TIM

APPLICATION LAYER

DATA ANALYTICS LAYER

DATA PLANE      CONTROL PLANE

TEDS DECODER

UUID PROCESSOR

PACKET DECODER

SnaaS LAYER

DB

Fig. 1: **SNaaS architecture and workflow.**

acts as the gateway between the TIM and the external network. TEDS can reside in TIM's memory, or at a central repository that is accessible by the NCAP from the network (referred to as Virtual TEDS). The standard mandates the following four TEDS sections.

• **Meta TEDS.** It contains referencing codes (for uniquely identification), different timing parameters (to detect the responsiveness of the TIM), and other information pertaining to the available transducer channels.

• **TransducerChannel TEDS.** To enable proper functioning of the addressed transducer channel, various operational parameters (such as type, physical units of measurements, operating ranges, measure of various delays, etc.,) are specified in this TED.

• **User's Transducer Name TEDS.** It contains the system referenced name (such as the model number and other manufacturer parameters) of the transducer.

• **PHY TEDS.** It contains information about the physical connection between the TIM and the NCAP.

The other TEDS section such as the calibration TEDS, Transfer Function TEDS, Frequency Response TEDs, etc., are optional; however, they can be used to increase the richness of the self-descriptive features of the transducer.

In the following section, we present the architecture of SNaaS, and use IEEE 1451 with certain design changes for IoT systems.

### B. System architecture

The SNaaS plug-n-play system architecture consist of four components: TEDS Service, Lightweight Directory Access Protocol (LDAP), NCAP and TIM (Fig. 1).

• **TEDS service and LDAP.** The online TEDS service is a web application that is used to generate TEDS for TIMs, and comprises of two modules: (1) TEDS Creator (TC) and (2) TEDS Generator (TEDS-G). The manufacturer is required to enter this information *once* for registering the device.

Manufacturers can use the TC to generate a TEDS e-form template with fields related to the configuration parameters of the TIM. TEDS-G loads the e-form template, and allows the manufacturers to include details about the TIM. On submission, an XML file with the TIM description is generated and is provided to the TEDS Encoder (TEDS-E), which encodes the TIM parameters into a Type-Length-Value (TLV) based binary (as per the IEEE 1451 standard TEDS data block representation). The encoded TEDS binary is stored in the LDAP directory service with the TIM UUID as the key.

• **TIM and NCAP.** TIMs with larger memory, communication bandwidth and continuous power source can store the respective TEDS in their memory, and it can be received by the NCAP on request. However, for resource constrained TIMs, the idea of virtual TEDS can be used for enabling self-describing capability.

SNaaS, a service layer that offers a conduit for streaming data from distributed physical sensors, is implemented in the NCAP for plug-n-play functionality. NCAP listens on the specific medium of communication for TIM association. The Packet Decoder (PD) parses the communicated information packet, following which the UUID processor checks with the local cache layer for TEDS history about the corresponding

TABLE II: Modulewise latency for Linux based NCAP

| Module | Case A ($\mu$s) | Case B ($\mu$s) |
|---|---|---|
| Packet Decoder | 1300 | 1300 |
| UUID Processor | 2 | 2 |
| Local Cache | 3300 | 1500 |
| LDAP Query | 0 | 897000 |
| TEDS Decoder | 3 | 3 |
| **Total** | **4600** | **900000** |

TIM. If the information is not available locally, the LDAP server is queried with TEDS UUID as the key to obtain the binary TEDS followed by an update to the local cache. The decoded TEDS binary is passed to the control plane for performing the necessary transducer channel configurations. Henceforth, the data plane provides the necessary abstraction for streaming the sensed information.

### C. Microbenchmarks

We have developed the SNaaS service layer for Android and Linux based NCAPs. It was tested and evaluated with two types of TIMs: (a) custom designed CC2540*Cheep* and (b) Arduino. CC2540*Cheep* uses Bluetooth Low Energy (BLE) as the communication primitive with the Android NCAP, while Arduino uses a USB Type B wired communication mode with the Linux based NCAP. Each of these TIMs were configured to send packets (containing the device UUID) every 100 msec.

Our preliminary microbenchmarks on latency measurements (for 50 iterations) for the linux NCAP are presented in Table II. We considered two cases for this analysis.

• Case A: binary TEDS is *available* in the local cache

• Case B: binary TEDS is *unavailable* in the local cache

For Case A, since the binary TEDS resides in the local cache, the total latency is only 4600 $\mu$s. For Case B, the time taken for fetching the binary TEDS from the LDAP server, storing it in the local cache and decoding takes 900,000 $\mu$s. It is to be noted that the LDAP is queried only *once* for a particular TIM, after which the local cache of the NCAP is updated with the respective history.

### IV. CONCLUSION

Interoperability is the key to manage systems of systems and to open markets to competitive solutions in IoT. In this paper, we addressed the device-level interoperability problem prevalent in current IoT-based solutions for smart cities. We proposed a SNaaS service layer, as part of a high level reference IoT architecture, that enables the creation of the plug-n-play infrastructure across platforms from multiple device vendors and implement it using the self-describing feature outlined in the existing IEEE 1451 standard. Such a SNaaS layer enables transparent management of sensors and collection of observational data, and depending on the domain, the layer can be implemented using existing standards from IEEE or W3C.

## REFERENCES

[1] "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats," *IEEE Std 1451.0-2007*, pp. 1–335, Sept 2007.

[2] Q. Zhao, Y. Simmhan, and V. Prasanna, "Incorporating semantic knowledge into stream processing for smart grid applications," in *International Semantic Web Conference*, 2012, pp. 1–16.

[3] S. Aman, Y. Simmhan, and V. Prasanna, "Improving energy use forecast for campus micro-grids using indirect indicators," in *International Workshop on Domain Driven Data Mining*, 2011, pp. 1–9.

[4] N. Govindarajan, Y. Simmhan, N. Jamadagni, and P. Misra, "Event processing across edge and the cloud for internet of things applications," in *20th International Conference on Management of Data*, 2014, pp. 101–104.

[5] P. Misra, Y. Simmhan, and J. Warrior, "Towards a Practical Architecture for the Next Generation Internet of Things," Tech. Rep., http://arxiv.org/abs/1502.00797.

[6] "Towards a Practical Architecture for Internet of Things: An India-centric View," http://goo.gl/DDNUkr.