SCALE: Safe Community Awareness and Alerting Leveraging the Internet of Things

The authors propose the Safe Community Awareness and Alerting Network (SCALE), a cyber-physical system (CPS) leveraging the pervasive Internet of Things (IoT) to extend a smarter, safer home to all residents at a low incremental cost. SCALE uses novel networking technologies, commodity sensor devices, cloud services, and middleware abstractions to sense, analyze, and act on sensed events in a distributed manner.

Kyle Benson, Charles Fracchia, Guoxi Wang, Qiuxi Zhu, Serene Almomen, John Cohn, Luke D'Arcy, Daniel Hoffman, Matthew Makai, Julien Stamatakis, and Nalini Venkatasubramanian

COMMUNICATIONS STANDARDS

Abstract

We propose the Safe Community Awareness and Alerting Network (SCALE), a cyber-physical system (CPS) leveraging the pervasive Internet of Things (IoT) to extend a smarter, safer home to all residents at a low incremental cost. SCALE uses novel networking technologies, commodity sensor devices, cloud services, and middleware abstractions to sense, analyze, and act on sensed events in a distributed manner. It monitors environmental factors (i.e. smoke, explosive gas) and automatically alerts residents via phone upon discovery of a possible emergency, enabling them to confirm the event and contact emergency dispatchers with minimal effort. This article describes the inception, design, development, and deployment of a prototype system to achieve these goals. We discuss lessons learned and future directions for general CPS/IoT platforms.

Introduction

With the increasing pervasiveness of computers in our daily lives, the Internet of Things (IoT) concept is transitioning from a future prediction to real-world deployments. With this manifestation comes a myriad of possible applications, from manipulating devices in our homes to large-scale automation of industries and public utilities. A common human-facing aspect of each of these applications is that they aim to improve our quality of life through inexpensive, commonly available technology. While home security systems have existed for decades, they are rather expensive services, and only in recent years have we seen components become cheap and available enough that hobbyists experiment with do-it-yourself systems. So it seems natural that an open system, made possible with these recent advances, should be created to improve the lives of under served populations that previously could not afford such advanced home security and safety monitoring systems. This motivation led our team, assembled in response to the SmartAmerica Challenge,[1] to envision, design, build, and demonstrate the Safe Community Awareness and Alerting Network (SCALE).

SCALE aims to improve the safety of residents through the use of modern connected devices and computer systems, particularly lower-income and elderly residents who often do not have access to advanced technologies such as home security systems, smartphones, and computers with Internet connections. To accomplish this goal, we designed an event-driven distributed system to sense safety-related data from devices in homes or on individuals, analyze it locally or within the cloud to detect possible emergency events, and automatically contact individuals (e.g. homeowners, caretakers, even emergency dispatchers) to notify them and confirm if there is indeed an emergency. We implemented a prototype of this system and deployed it in Montgomery County, Maryland, USA to enable rapid integration of components and testbeds from different partners.

The immediate goals of the SCALE project are:

- Demonstrate our ability to extend a connected safe home to everyone at a low incremental cost.
- Jump-start a live testbed for identifying and researching IoT challenges (e.g. middleware, networking, etc.).
  - Identify suitable sensors, data schemas, and algorithms for detecting possible emergency events.
  - Implement and test workflows for cloud-based analytics and alerting.
- Demonstrate an open data platform for connecting disparate systems with minimal coordination.

System Architecture

SCALE devices upload sensed events, and the analytics service looks for possible emergencies, it sends residents emergency alerts to confirm or reject, and interested individuals (i.e. emergency dispatchers) visualize events through a dashboard. This section discusses the high-level requirements, logical components, architectural design decisions, and implementation details of the system prototype. It first discusses a *cloud data exchange* and then the components of the system that perform *sensing*, *analysis*, and *actuation*.

Cloud Data Exchange for IoT

To facilitate machine-to-machine (m2m) communication for exchanging IoT data in SCALE (sensed events, analytics, alerts, etc.), we propose the Data in Motion Exchange (DIME) system, shown in Fig. 1. We envisioned DIME as an open communications hub for IoT that simplifies the development and deployment processes.

DIME allows any device or service to publish or subscribe to any other data feed, regardless of the protocols used at the device level. This simple loose coupling enables developers to incorporate new services and devices without the need to modify existing ones. This simplifies system evolution, and it also creates a level playing field for innovation. Any party can introduce new

Kyle Benson, Guoxi Wang, Qiuxi Zhu, and Nalini Venkatasubramanian are with University of California.

Charles Fracchia is with BioBright.

Serene Almomen and Julien Stamatakis are with Senseware, Inc.

John Cohn is with IBM.

Luke D'Arcy is with Sigfox.

Daniel Hoffman is with Montgomery County.

Matthew Makai is with Twilio.

[1] http://smartamerica.org/

 0163-6804/15/$25.00

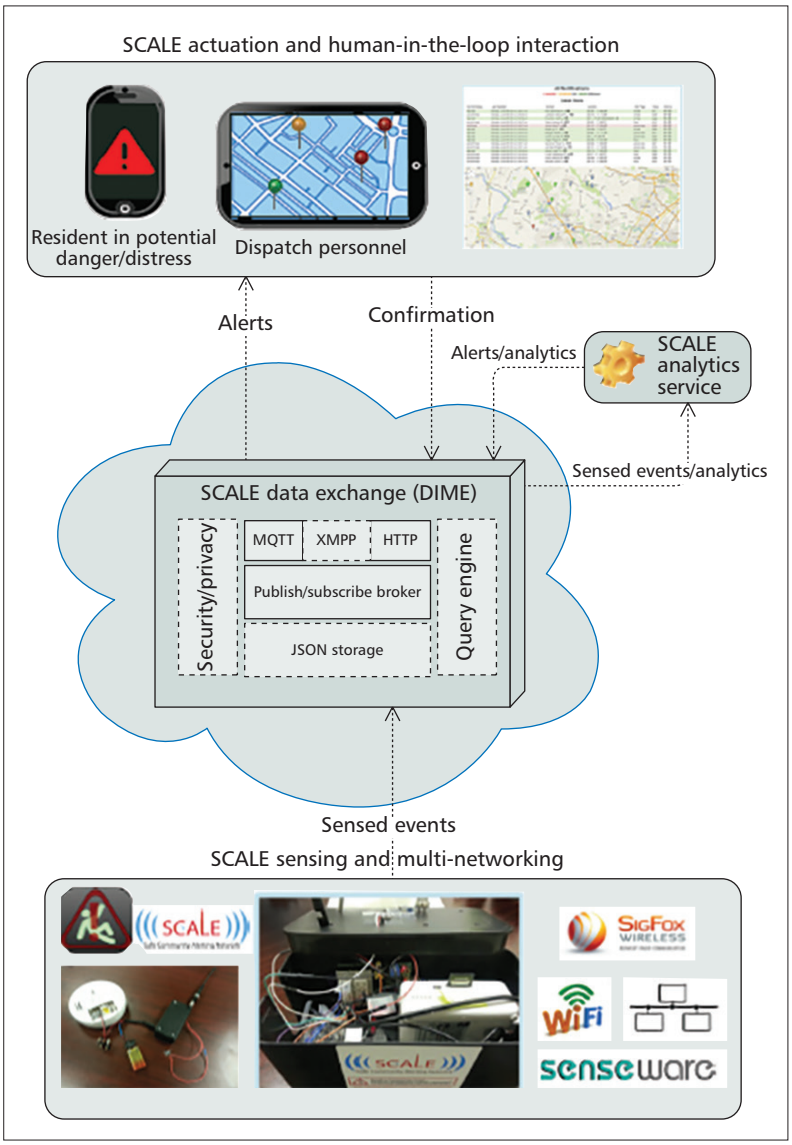

Figure 1. DIME facilitates the exchange of data between main SCALE components. DIME Components shown in solid boxes have been implemented, and those in dashed boxes remain as future work.

capabilities, or improvements to existing ones, to the system with minimal need for coordination among current components. They can perform analysis on sensed data, or even higher-level events, and contribute the results back to the exchange, driving science and innovation faster as more devices connect.

Sensed Event Data: To build an exchange for IoT data, we first defined the type of data that DIME should handle. We decided to treat raw sensed data and higher-level events equivalently. This aligns with our concept of *virtual sensors*, previously proposed in [1]. *Virtual sensors* abstract low-level data by processing sensor data streams, which may be directly or indirectly derived from physical sensor devices, and exposing higher-level semantics through advanced analytics.

Recognizing the rich amount of information contained within a higher level event as well as subtle device differences that affect lower-level events, we wanted a well-adopted flexible schema that could allow, but not require, inclusion of additional information fields beyond what is necessary to convey the sensor reading. These additional fields should not break the schema or require all entities in the system to understand them.

For simplicity and flexibility, we opted to use JSON to format the data for transmission to the broker, as it provides a commonly-used self-describing format supported by mature software modules. We defined what we thought was a reasonable starting point for the schema. It includes information about the platform (hardware, operating system, etc.), sensor (device type, identifier, etc.), data (units, value, timestamp, priority, etc.), a pointer to the specific schema in use to facilitate interoperability, and any other miscellaneous domain-specific information that developers want to include. One should note that we do not believe this schema to be comprehensive; rather, we envision a system where different domains could define their own schema and publish information about how to interpret it so as to encourage interoperability between vendors/systems.

Current Implementation: In its current form, DIME uses MQTT,[2] a fast, lightweight, publish-subscribe-style protocol. It was developed by IBM for lightweight telemetry, donated to open source, and has since gained popularity for use as an m2m protocol for IoT data. The publish-subscribe model allows multiple servers to collect data from DIME and multiple clients to send it without requiring any configuration on our part. The DIME server currently uses the open source Eclipse Paho MQTT broker.[3] While Paho could be run anywhere, we used IBM's MessageSight[4] software appliance, which handles millions of concurrent data streams, running on the IBM SoftLayer Cloud.

In DIME, sensor data is published to a particular topic, which consists mainly of a device identifier and sensed event type. Other services, such as the SCALE server, subscribe to this data by a particular device, sensor type, or just to all data.

For compatibility, DIME also provides a RESTful interface, implemented via HTTP, initially residing on the SCALE server for ease of deployment. This interface translates incoming data into the proper format and publishes them via MQTT. In this manner, we quickly implemented DIME as a simple MQTT server, though we plan to extend it to directly support other protocols (e.g. HTTP and XMPP).

Sensing

This section describes the development and deployment of several SCALE clients that sense, minimally analyze, and report data to DIME for ingestion by the analytics server.

Networking Technologies: To support a heterogeneous mix of devices and improve the client's flexibility in deployment, we integrated multiple networking technologies. In addition to the standard Wi-Fi and Ethernet connections, the clients supported ultra-narrowband (UNB) wireless adapters. UNB allows for long-range, low-power, low-bandwidth uplinks. Sigfox provided a UNB basestation to install in Montgomery County. We

[2] http://mqtt.org.

[3] http://www.eclipse.org/paho/

[4] http://www-03.ibm.com/software/products/en/messagesight

were able to deploy several SCALE devices with Sigfox UNB adapters in Rockville, Maryland, and send data to DIME via the basestation from up to several kilometers away, despite using lower-powered basestation and client adapter antennas.

Sigfox adapters send data in 12-byte packets, so MQTT was not an option. Instead, we coded the data to fit within this packet and created the aforementioned HTTP interface where Sigfox directed this data. We also integrated Senseware's proprietary mesh networking solution into the SCALE system, as described below.

Hardware Platforms: We wanted a flexible client platform to allow deploying heterogeneous sensors, devices, and networking technologies. Some clients may plug into a stable power source and Internet access to support a multitude of sensors and more advanced local data analytics, while others may be battery-powered and just upload raw sensed data via wireless. To support the pervasiveness of these systems and address the latter of these device types by reducing reliance on home Internet access, crucial for our mission to support under served populations, we aimed to integrate platforms and technologies that could provide long-range low-power connections. To address both styles, we chose to use commodity off-the-shelf components wherever possible, which had additional benefits of reducing infrastructure costs; increasing the number of possible integrated devices and sensors; reducing development costs by leveraging extensive community support; and allowing other researchers, hobbyists, and new team members to easily understand our design so that they may copy and extend it.

We first built a general-purpose sensor box named FlexSCALE that supports many different sensors and network adapters. The compute units and sensors are housed within a large cable box to protect wires and maintain a cleaner facade. Environmental sensors (e.g. light and temperature) were fastened on top so they protruded from holes in the lid, gaining external access with minimal wiring exposed. The initial version housed both a Raspberry Pi and a Sheevaplug, each running a form of Debian Linux, as the compute units. We transitioned to just using the Raspberry Pi to simplify platform support and handle a greater variety of peripherals thanks to I/O ports and pins other than USB and Ethernet.

Each FlexScale box has light (luminance), explosive gas, passive infrared (motion detection) sensors, an accelerometer (acting as a seismograph), and thermometer as well as a Wi-Fi dongle and a Sigfox UNB adapter. A powered USB hub supported the two USB sensors and two USB network adapters on the Sheevaplug and older Raspberry Pis.

In contrast with the larger and more extensible FlexSCALE Box, we experimented with dedicated devices to monitor a single sensor and report its readings with almost no analysis. We were particularly interested in retrofitting existing household sensing devices and connecting them with the SCALE service. Therefore, we modified an off-the-shelf 9-volt smoke detector and attached it to an Arduino Micro for the purpose of monitoring the voltage level of the battery. See Fig. 2 for the wiring diagram used for this device. The Arduino constantly sends (every ~4 sec.) the measured voltage level to DIME via a Sigfox adapter. If this level drops significantly, indicative of the alarm going off, the server sends an alert. The theory here is that the alarm consumes more power than just the sensor itself and so the additional function drops the voltage level of the battery significantly. The Arduino and Sigfox devices fit into a small project box, similar in size to a mint tin.

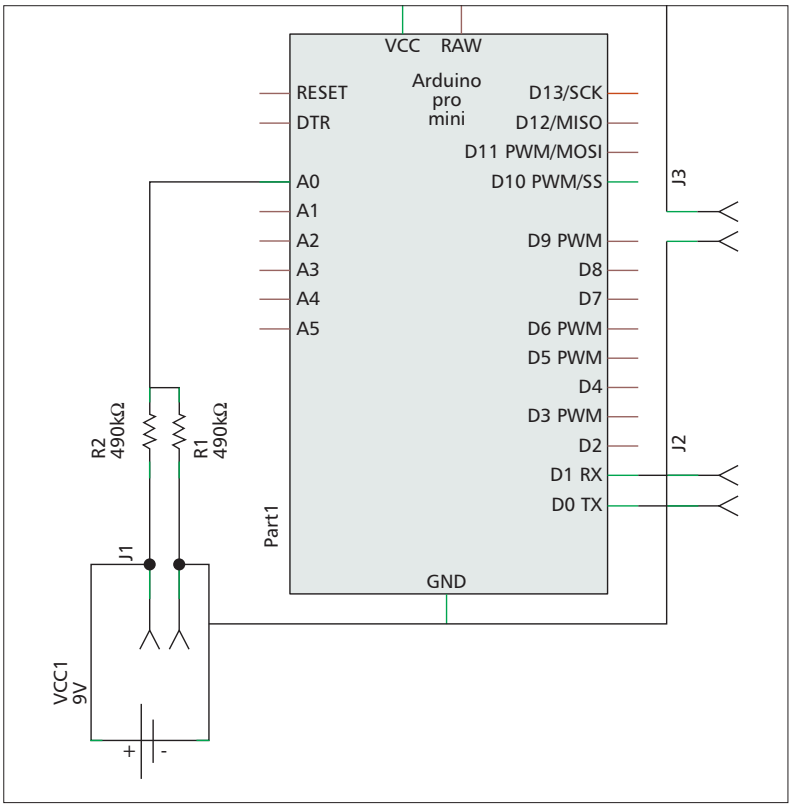

Figure 2. Wiring diagram for the hacked smoke detector device.

To complement the aforementioned dedicated and flexible sensing platforms, we also built an Android application for personal fall detection. It analyzes the device's accelerometer readings using the algorithm presented in [2]. Upon detecting a user falling, the application presents them with an option to cancel the alert, thus preventing false alarms, before a countdown timer expires and the phone publishes the alert via MQTT to call for help.

To test and showcase how existing proprietary systems could integrate with DIME and SCALE with minimal modifications, we partnered with Senseware, a Virginia-based startup. They build modular sensor devices that transmit data via mesh networks to a gateway for upload to a web-based cloud service. The user-friendly devices are easy to deploy and can have a variety of connected sensors (i.e. air quality, humidity), making them an ideal candidate to expand the SCALE testbed with commercial hardware. Senseware integrated their sensors' data by forwarding it to a Senseware-specific HTTP endpoint to facilitate this connection, similar to how we integrated Sigfox devices.

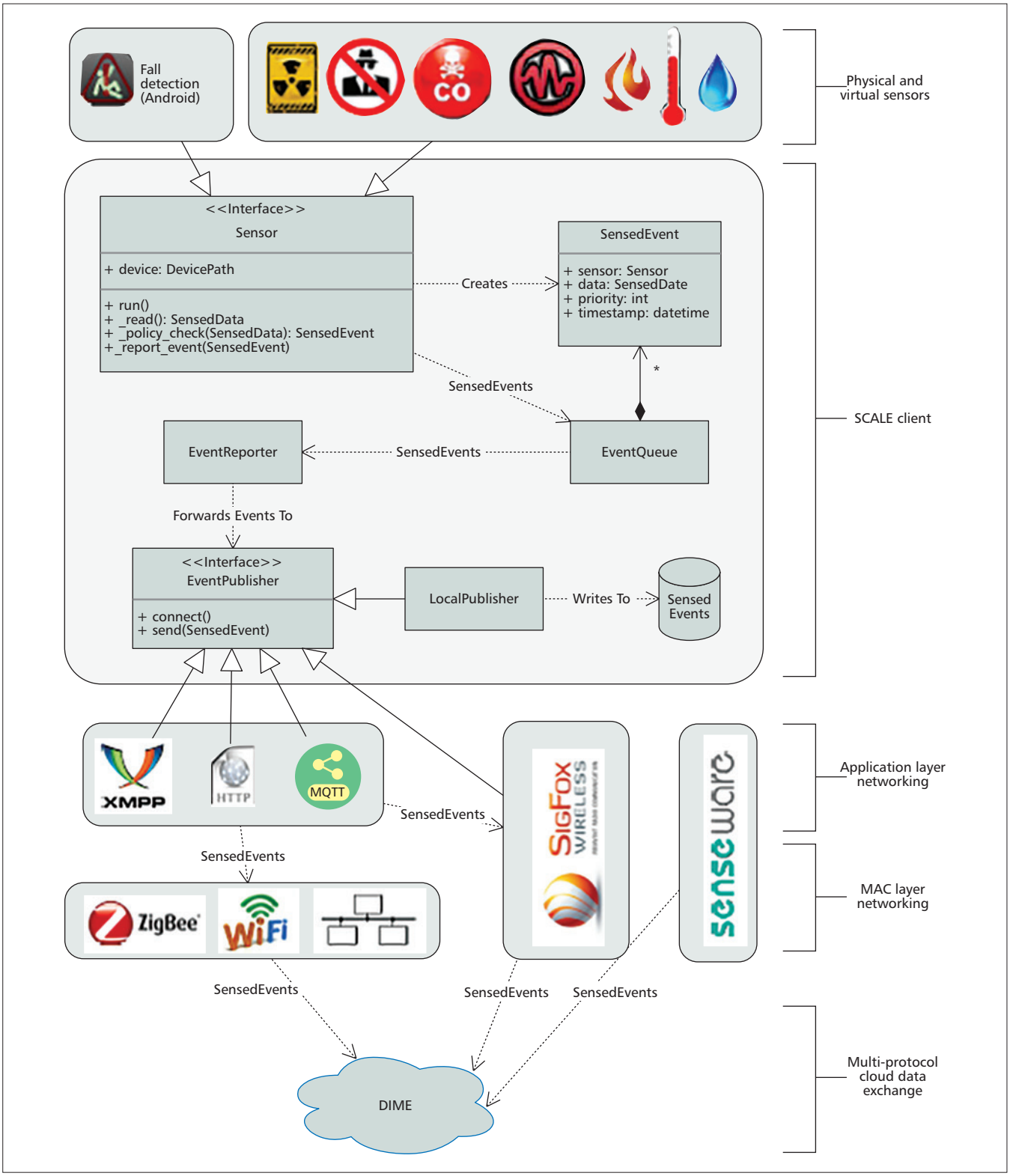

Figure 3. The SCALE Client architecture.

Software Design: We wanted a cross-platform extensible software package that runs on the majority of devices. This package should be modular and support plugging in different component implementations (e.g. new sensors or network protocols) without disrupting other modules. Adding or changing hardware components should not require any software changes, but should rather be handled through a simple configuration file.

Figure 3 shows the prototype FlexSCALE software we built to address the above requirements. This Python package connects with various sensor devices attached to the compute system, records data, and publishes events according to some policy. Data originates at an

instantiation of the abstract sensor class, which allows us to rapidly connect new sensor types and define new *virtual sensors*. Sensors create SensedEvents, which encapsulate the sensor data schema described earlier, and place them in a queue for reporting to DIME or further analysis by relevant VirtualSensors.

Each networking protocol that connects the client to DIME is abstracted with a concrete instantiation of the EventPublisher class. Similar to adding new sensors, this allows us to easily add new protocols and API endpoints with minimal additional code. It currently supports MQTT via Wi-Fi or Ethernet, Sigfox ultra-narrowband (UNB), and local storage. EventPublishers also provide a degree of control over quality of service (QoS), currently just in the form of transmitting higher-priority events first. We added this feature early on to address the UNB transmitters' low bandwidth.

We used SaltStack[5] for configuration management: remotely deploying and updating software on the sensor boxes. We chose SaltStack because it is highly scalable, supports redundant master servers, and (most importantly) connects with devices deployed behind network address translators (NATs) as are commonly found in residential homes.

Analytics

The SCALE analytics service monitors sensor data and events streaming from DIME and publishes detected emergency events, which may trigger alerts to individuals when appropriate as described earlier. Refer to Fig. 4 and the description below for how we designed and implemented the analytics server.

We implemented the analytics engine as an asynchronous event-driven Python server that acts on sensed events in accordance with their type using appropriate event-handlers. Thus, adding new sensor and event types only requires additional programming by end application developers, not those responsible for server development.

The server, deployed on the IBM BlueMix platform, receives sensed data through Eclipse Paho's MQTT client[6] and routes it to the appropriate event-detection function. These functions, which we refer to as *virtual sensors*, convert lower-level events to higher-level ones (e.g. alarm buzzing to smoke detected to possible fire), escalating events and publishing them back to DIME. When a possible emergency event is detected, SCALE may alert a resident as described earlier.

We used the above incremental approach as it allows different server components to live on or replicate across separate machines and locations, improving scalability, response times, modularity, reliability, and ease of creating an audit trail. An audit trail exposes intermediate events to external entities, which helps in building trust in particular event sources (i.e. sensing devices, event-detection algorithms) and adding new hooks for separate services to make use of these states. We accomplished this distributed approach using the Celery task queue manager[7] that distributes event-handling across worker processes.

Some historical storage of recent events is necessary to detect changes over time and disambiguate sensor readings indicative of the same event. We used the Django framework's object-relational mapping (ORM) to abstract the PostgreSQL database tables seen in Fig. 4 as Python objects. Periodically, the database removes old events, though in the future we will instead archive them for historical analysis and audit purposes.

Actuation

This subsection describes SCALE's mechanisms for interacting with and alerting human users.

Alerting: Once possible emergency events are detected, concerned individuals must be notified in a timely, reliable, accessible, and interactive manner. Users will receive alert messages after connecting their home monitoring devices to SCALE and registering these devices and contact information with the alerting system. Ideally, this system could eventually integrate with emergency dispatch centers to automatically alert authorities. To mitigate false-positives, it supports a confirmation step in which the user determines whether the emergency is real and emergency personnel should be alerted.

Because SCALE especially aims to make the system as accessible as possible, especially for

[5] http://www.saltstack.com/

[6] http://www.eclipse.org/paho/

[7] http://www.celeryproject.org

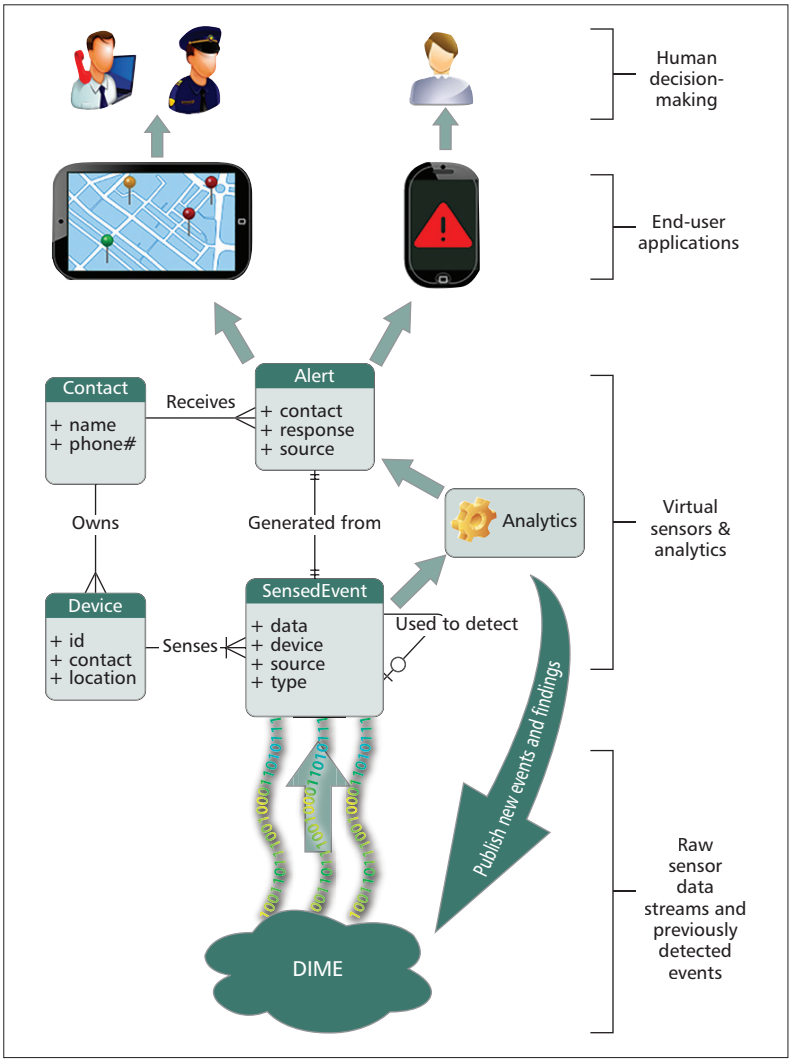

Figure 4. Depiction of data flow and database tables in the analytics and alerting services.

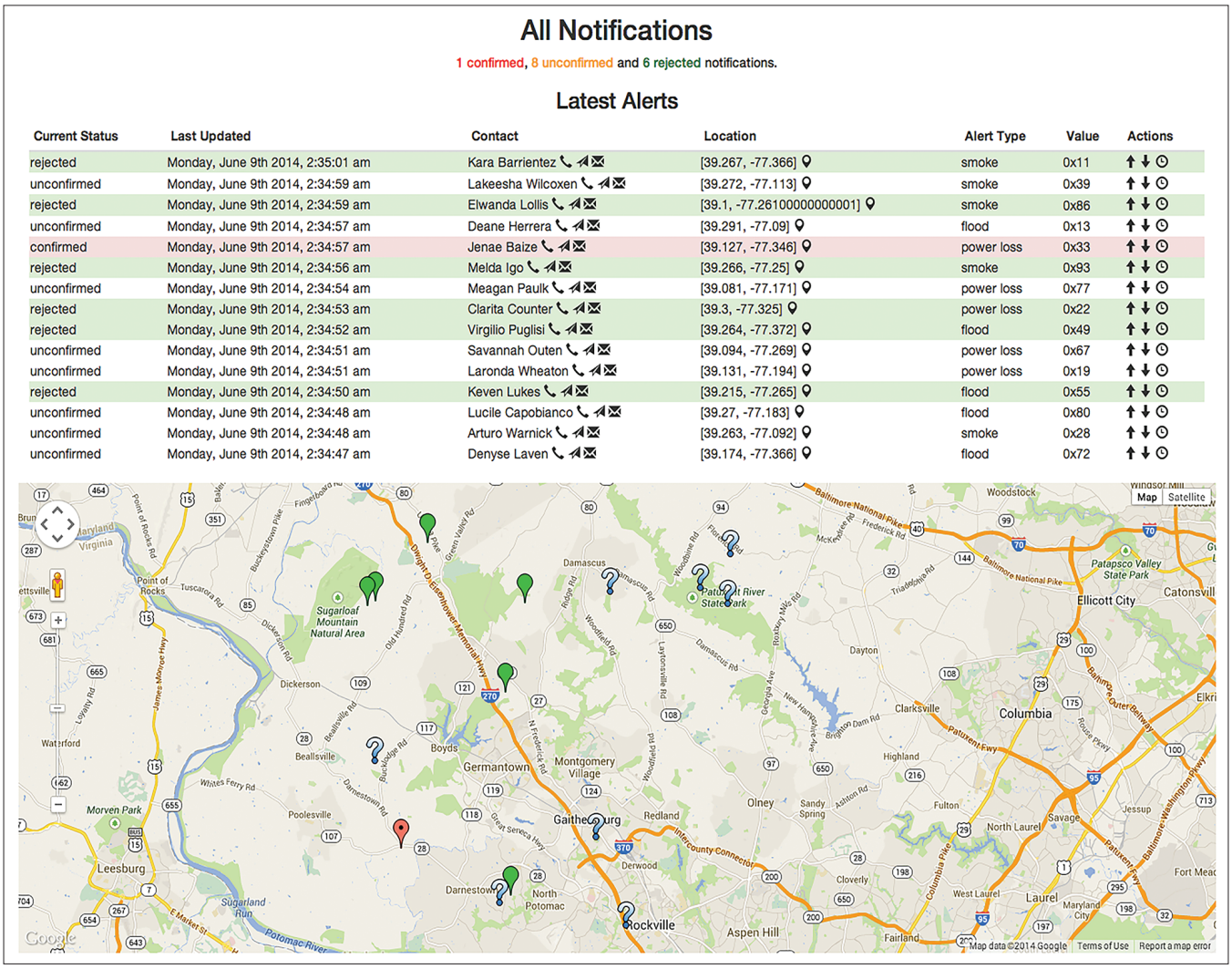

Figure 5. The main view of the SCALE dashboard.

lower-income and/or less technologically-savvy users, it does not require access to a computer or smartphone when receiving and acting on alerts. It supports simple phone calls so that users with land lines, but no cell phones, can still use it. It does support SMS (text messaging) as most people in the U.S. nowadays have cell phones, especially since government programs[8] exist to provide them to low-income residents.

While a smartphone application is a potential future addition, we opted to use an Internet telephone service for alerting. We chose Twilio, which has a rich API for programming interactions with users through the server's web interface, to issue SMS/phone call alert messages and handle correspondence with participants (event confirmation/rejection, registration/unregistration, contact method preferences, etc.).

When the analytics subsystem detects a possible emergency, it sends an Alert message through MQTT to instruct the alerting subsystem to contact the registered user(s) of the device from which the sensor data originated. This contact info is retrieved from the database as shown in Fig. 4, and the database stores an Alert entry representing this communication. When the contact responds, the database updates the state of the Alert to *rejected* if the user responds with "emergency" or presses 1 and *event confirmed* if the user responds with "okay" or presses 2.). If no one responds within some amount of time (currently 30 to 60 seconds) of initiating the alert, a trigger fires that escalates the emergency event. Currently, it is set to confirm the event, but public officials likely would adopt a different policy that perhaps dispatches an individual to investigate further rather than scrambling an entire unit.

Dashboard: To help dispatch personnel visualize alert events and sensed data, we built a dashboard for the SCALE system. We wanted an intuitive, lightweight, web-based solution so we could later port it to mobile devices and borrow functionality for a smartphone application aimed at residents for monitoring their personal SCALE deployment(s). Figure 5 shows the main user interface.

The main view of the SCALE dashboard presents a list of recent alerts, their locations in a Google Maps view, and a summary of the number of high, medium, and low priority events. It includes contact information about the individual alerts and a currently non-functional interface for calling,

[8] http://www.fcc.gov/lifeline

texting, or emailing residents. The user can also confirm or reject events manually. A second view presents raw sensed events as they arrive, which is simply for debugging purposes. The dashboard, also hosted on BlueMix, is built on top of software designed by BioBright. It is written using Node.js at the backend, Javascript and Twitter Bootstrap in the front-end, and a browser MQTT client.

Conclusions and Future Research

Our experience in designing, developing, and deploying the first iteration of SCALE described in this article has proven the feasibility of a distributed IoT approach to improving resident safety at a low incremental cost. This initial exploration requires much further development before this system could be deployed in any real capacity. However, the lessons we learned will help drive future research and development for IoT. We discuss some of these topics below and present our future plans and vision for SCALE.

Resilience Concerns

From a hardware perspective, our biggest lesson learned was that *cheap sensors break*. We purchased many of our devices online for under $10 to $20 (U.S.) and some failed. The explosive gas sensors in particular tended to burn out after a few uses. While some of these issues can be alleviated by using better quality components, this likely drives up the price of the device without completely ensuring reliable operation, and so care must be taken to plan for these issues.

Regarding power, we found that the number of peripherals on the Raspberry Pi required a higher-amperage power adapter. We also used a powered USB hub to support all of the USB sensors and wireless adapters used on the Sheevaplug. We experimented with battery backup and determined that the system dies after about 10 hours. Future work will explore graceful degradation of devices (turning off network adapters, adjusting sampling rates, etc.) to improve this battery life.

From a software perspective, we found the design of the client around simple abstract pipeline components to be very flexible when adding new hardware support.

Currently, we are experimenting with additional networked sensor devices to extend the coverage of a SCALE deployment and improve its resilience. We are adding support for an ad-hoc Wi-Fi mode that supports distributed emergency detection and alerting even during power and network failures by having FlexSCALE boxes exchange data with each other directly. We are also adding inexpensive battery-powered microcontrollers with attached sensors and IEEE 802.15.4-based wireless so additional sensors can be deployed throughout a residence without requiring additional FlexSCALE boxes.

Data Exchange

While we found MQTT suitable for rapidly developing an IoT system, we did find it limited due to its simplistic lightweight approach. Below we outline some considerations for future IoT protocol standardization efforts and security considerations for data management in IoT systems.

Standards Considerations: When designing our analytics system and topic hierarchies, we found MQTT's lack of support for fine-grained queries somewhat limiting. It does not handle ranges at all, and the expressiveness of wildcards cannot match that of regular expressions. For example, to perform a query over a target geography one would need to define a tag for that geography, which limits flexibility for defining new targets. Our current inefficient solution is to subscribe to all events and filter them based on content. Future protocols should consider the desire to issue such queries and filters, as sorting through the results by content on the client-side may be intractable with the larger-scale systems the IoT vision promises.

A major advantage of MQTT is its lightweight nature and simplicity. Future IoT standards should follow this model, while allowing for extensions that provide additional services when, but only when, developers/deployers wish to use them. For example, the size of the DDS[9] standard may intimidate some newcomers, whereas getting started with MQTT takes only a matter of minutes. Protocol designers must keep in mind that many IoT developers will enter the market with little systems experience or come from a Web 2.0 background. As such, providing a simple intuitive starting point, perhaps with RESTful APIs, for them to develop systems will help lower the barrier to entry, resulting in more projects with diverse applications. This approach appears to have worked very well with Node.js, which has enjoyed rapid adoption in part because it gives the developer community the freedom to pick from a variety of options for accomplishing a given task rather than specifying one standard approach. To further lower this barrier, future standards should also allow developers to use familiar tools, languages, etc. whenever possible. For example, they should emphasize interoperability with other protocols, such as how CoAP [3] can interoperate with HTTP.

Security and Privacy: While we did not implement security mechanisms in SCALE beyond requiring SSH keys to remotely access devices, we did discuss security and privacy implications throughout the project and plan to address them in future versions. MQTT supports authentication and identification directly, but not authorization. It can be run using TLS so that the user name and password used for authentication are encrypted during transmission. Identification is handled using a unique identifier or a public digital certificate, with the latter clearly involving management of keys. Some MQTT server implementations provide authorization as an added service. In a scenario where user privacy and integrity of the data and communications is crucial, such a server should be used. This allows the server to determine which client devices have access to which resources, i.e. which topics they are allowed to publish and subscribe to. This would prevent unauthorized individuals from retrieving readings from devices they do not own, as well as prevent publishing of information to a topic representing a different device. However, this does not validate the actual data in question, which could still be faked by an individual with the proper secret keys.

Because SCALE especially aims to make the system as accessible as possible, especially for lower-income and/or less technologically-savvy users, it does not require access to a computer or smartphone when receiving and acting on alerts. It supports simple phone calls so that users with land lines, but no cell phones, can still use it.

[9] http://portals.omg.org/dds/

One open concern is that of the devices' physical security. As they are located in residents' homes they could be physically tampered with, moved, or have their code modified by knowledgeable users. This could result in undefined behavior, misleading event reports, or completely spoofed data.

One open concern is that of the devices' physical security. As they are located in residents' homes, they could be physically tampered with, moved, or have their code modified by knowledgeable users. This could result in undefined behavior, misleading event reports, or completely spoofed data. This is one of the main reasons for involving human-in-the-loop sensing in order to confirm events before notifying emergency personnel. Whether this step is truly enough to ensure correctness of the data in question is a policy question outside the scope of this article.

Acknowledgments

This work was supported in part by the National Science Foundation award nos. CNS 1143705, CNS 0958520, and CNS 1450768. The authors would also like to thank NIST and the Presidential Innovation Fellows for issuing the SmartAmerica Challenge that initiated our team's creation; Beall's Grant Apartments for hosting our initial test deployment; Kevin Malby and Phu Nguyen from UCI for contributing to SCALE; Mani Chandy from Caltech and Sharad Mehrotra from UCI for their discussions involving sensing and SCALE; World Sensing, Anomaly Systems and the Community Seismic Network at Caltech for providing additional sensors.

References

[1] B. Hore *et al.*, "Design and Implementation of a Middleware for Sentient Spaces," ***2007 IEEE Intelligence and Security Informatics***, May 2007, pp. 137–44.
[2] A. Bourke, J. O'Brien, and G. Lyons, "Evaluation of a Threshold-Based Tri-Axial Accelerometer Fall Detection Algorithm," *Gait & Posture*, vol. 26, no. 2, 2007, pp. 194–99.
[3] Z. Shelby *et al.* RFC 7252 — The Constrained Application Protocol (CoAP).

Biographies

Kyle Benson (kebenson@uci.edu) is a computer science Ph.D. student at UC Irvine and NSF GRFP Honorable Mention. He researches resilient pervasive sensing communications platforms leveraging low-cost Internet-connected devices. During the SmartAmerica Challenge, he led development on SCALE. His current research focus is on the use of geo-aware overlays to improve IoT communications during disaster scenarios, thereby enhancing situational awareness and emergency response efforts.

Charles Fracchia is an IBM Ph.D. Fellow at the MIT Media Lab in Joe Jacobson's Molecular Machines group, and in the Church lab at the Wyss Institute at Harvard Medical School. Charles is a founder of BioBright, a company building a 'smart lab' user interface that can capture and track everything that happens in a biological experiment.

Guoxi Wang is a master's student at the University of California, Irvine, majoring in networked systems. He received his B.S. in information security from Wuhan University in 2012. His research interests include wireless and sensor networks and software-defined networking.

Qiuxi Zhu is a Ph.D. student in the Department of Computer Science at the University of California, Irvine. He received his B.E. degree in automation from Zhejiang University in 2013. His research interests include mobile sensing, mobile networking, and Internet of Things.

Serene Almomen is CEO and co-founder of Senseware, whose cloud-based data platform, wireless modular technology, and patent pending universal sensor interface provides clients with real-time data to optimize facility performance, meet regulatory requirements, and reduce costs. Serene is passionate about using technology to improve the world around us.

John Cohn is an IBM and IEEE Fellow in the IBM Internet of Things Division. He received a BSEE from MIT, and a Ph.D. in CE from Carnegie Mellon. John is eager to share his love of science and technology with anyone who will listen.

Luke D'Arcy is working to build a SIGFOX network covering the entire U.S. Previously he was a founder of Neul, an IoT networking company that was bought last year by Huawei, and of the Weightless SIG, a standards body defining a standard for low power IoT networks. Before that he was one of the first full time marketing staff at the chip company CSR.

Daniel Hoffman is the first chief innovation officer for Montgomery County, Maryland, a position he has held since October 2012. The program he oversees serves as a laboratory for civic improvement and a safe place to test out new processes, technologies, and ideas. Project topics vary from the Internet of Things (IoT) to autism technology to food security and more.

Matthew Makai is a Twilio developer evangelist and software developer with an affinity for Python. He was a speaker at EuroPython, DjangoCon US in 2014, and PyCon in 2015. Matt also writes Full Stack Python, which helps more than 25,000 developers a month learn to build and deploy Python web applications.

Julien Stamatakis is CTO and co-founder of Senseware, whose cloud-based data platform, wireless modular technology, and patent pending universal sensor interface provides clients with real-time data to optimize facility performance, meet regulatory requirements, and reduce costs. Julien is an award winning electrical engineer who designed and developed the mechanics behind Senseware.

Nalini Venkatasubramanian is a professor in the School of Information and Computer Science at the University of California Irvine. She has significant research and industry experience in the areas of distributed systems, adaptive middleware, pervasive and mobile computing, cyberphysical systems, distributed multimedia and formal methods, She has more thn 200 publications in these areas.