Making P-Space Smart: Integrating IoT Technologies in a Multi-office Environment

Orestis Akribopoulos, Dimitrios Amaxilatis, Vasileios Georgitzikis, Marios Logaras, Vasileios Keramidas, Konstantinos Kontodimas, Evangelos Lagoudianakis, Nikolaos Nikoloutsakos, Vasileios Papoutsakis, Ioannis Prevezanos, Georgios Pyrgeris, Stylianos Tsampas, Vasileios Voutsas, and Ioannis Chatzigiannakis

Computer Technology Institute & Press, and
Computer Engineering and Informatics Department, University of Patras
{akribopo,amaxilatis,tzikis,logaras,ichatz}@cti.gr,
{keramidas,kontodimas,lagoudiana,nikoloutsa,papoutsaki,prevezan,
pyrgeris,tsampas,voutsas}@ceid.upatras.gr

Abstract. Internet of Things technologies are considered the next big step in Smart Building installations. Although such technologies have been widely studied in simulation and experimental scenarios it is not so obvious how problems of real world installations should be dealt with. In this work we deploy IoT devices for sensing and control in a multi-office space and employ technologies such as CoAP, RESTful interfaces and Semantic Descriptions to integrate them with the Web. We report our research goals, the challenges we faced, the decisions we made and the experience gained from the design, deployment and operation of all the hardware and software components that compose our system.

Keywords: Internet of Things, Wireless Sensor Networks, Smart Buildings, Building Automation, CoAP.

1 Introduction

Internet of Things (IoT) refers to the integration of uniquely identifiable Smart Objects and web-based semantic entities and services via the Internet. Ultimately, IoT will offer abstractions for the sensor and actuator hardware and wireless networking technologies that will allow application developers to operate freely without worries for low level restrictions or limitations. Although a lot of research has been devoted towards this direction such abstractions for deploying and annotating sensor devices as smart objects is neither straightforward nor fully standardized. Thus application development for the IoT is still tied up to some negative characteristics of WSNs. Therefore it is still important to address fundamental problems, as described in [5], like:

– hardware, software and networking heterogeneity,
– intermittent connectivity,

- application scaling issues,
- simplification of the development and deployment cycle,
- absence of standardized service and capability descriptions,
- big data management

Research dedicated to solving the issues and facilitate large scale installations resulted in the design and development of some important software systems like *SenseWeb* [1], *GSN* [2], *sMAP* [6] and *Cosm*[1]. These systems focus on some of the problems mentioned above and provide scalable solutions for management of big data. Yet, further research and engineering work is required.

One major challenge that has not been addressed at adequate level is to provide bidirectional communication between IoT applications and smart objects in an abstract way. In IoT applications issues like monitoring and controlling devices and workplaces require the interaction of smart objects with data and services residing on the web. The ability to trigger actions in response to special events or situations is critical to achieve goals like minimizing energy requirements or adapting environmental conditions to user preferences.

In this work, we present how we employed IoT technologies and research results to address common everyday issues in a prototype system installed in a multi-office space located in Patras, Greece called P-Space. We deal with all the levels of the system: (a) design and installation of low level hardware devices, (b) wireless networking issues for interconnecting smart objects to the Web, (c) storage of the data generated to provide historical comparisons, and (d) high level interfaces to facilitate user interaction.

The smart objects deployed in P-Space are of our own design based on the Arduino open-source electronics prototyping platform[2]. Wireless networking is achieved by attaching XBee 802.15.4 modules[3] to the arduino boards. In particular we used the ATmega328 enabled Arduino Pro Mini to drive our control and sensing boards on objects like desks, lamps, doors and faucets. The resulting smart objects are wirelessly connected, have low cost and are compatible with a broad range of sensor and actuator components.

Communication for querying smart objects for sensor data and sending commands to actuate attached devices we used the CoAP protocol [12]. We incorporate in CoAP high level descriptions of the sensor and actuator devices and utilize CoAP extensions for providing notifications for events or sensor readings on the sensor devices.

We combine CoAP with Überdust[4][3], a brokerage web service for connecting smart objects to the Internet of Things, providing storage, sharing and discovery of real-time and historical data from smart objects, devices & building installations around the world via the Web. Überdust provides high-level language-independent APIs so IoT application developers may choose their favorite programming or scripting languages. Communication with Überdust is

[1] `https://cosm.com/`
[2] `http://arduino.cc/`
[3] `http://www.digi.com/xbee/`
[4] `https://github.com/Uberdust/webapp/wiki`

achieved using the Überdust RESTfull and Websocket APIs, offering an easy-to-use, resource-oriented model to expose services. Publishing and consuming functionalities are implemented using both the HTTP and CoAP `Get` and `Put` methods to represent data exchange. Überdust acts as our entry point to the smart objects network, providing important IoT functionalities for the interaction of end users with the smart objects mentioned above, like CoAP [12] , RESTful interfaces and Semantic descriptions.

Finally, we also implemented a number of IoT applications to provide control and browsing capabilities over the deployed platforms. The characteristics of our system allowed us to experiment with multiple programming and scripting languages based on the requirements of each application. We experimented and developed applications with Python, Java, JavaScript, PHP and even Bash Shell scripts to perform various tasks with the sensor and actuator capabilities of the whole installation.

In order to organize the smart objects and provide interfaces and methods for querying and controlling the devices in a natural way, we propose the abstraction of Virtual Node. The idea is to group real devices that match specific semantic description as unified semantic entities. Then the application developer as well as the user can used them as a single entity. Our experience indicates that this abstraction is very useful for improving the overall ability to maintain the system and develop applications.

The next Section presents a high level architecture of the system we designed, the targets set and the design choices we made. Section 3 presents the smart objects we designed, the requirements for each device, the issues we faced and the experience we gained. Section 4 presents the CoAP protocol we used for communication in all levels of our system. Next, in Sections 5&6 we present the user interfaces we implemented to facilitate interaction with users and finally in Section 7 we explain the experience we gained from this work, the major issues still to be solved and our future targets.

2 High Level System Architecture

We designed our system in 3 separate layers that communicate with each other by exchanging messages of commands or sensor readings. Figure 1 shows the 3 independent layers and their communication via RESTful and Websocket APIs offered by Überdust.

The bottom layer comprises of the hardware sensor boards we designed and attached to objects inside P-Space. This layer provides sensor readings over wireless communication to a central controller device that acts a translator between 802.11 frames and 802.15.4 packets.

Überdust is the nerve center of our system as it is responsible for connecting the low level hardware devices with the IoT applications and interfaces enabling the rapid development of IoT applications. Überdust is designed as a Machine-to-Machine system and its functionality is provided through a semantic-based approach to facilitate IoT application development. The most innovative feature

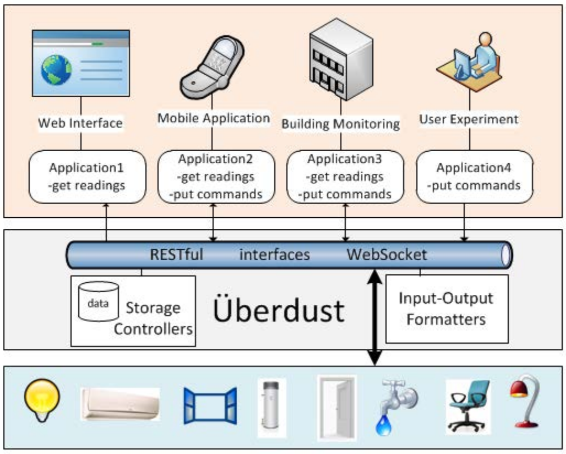

Web Interface
Mobile Application
Building Monitoring
User Experiment
Application1 -get readings
Application2 -get readings -put commands
Application3 -get readings -put commands
Application4 -put commands
RESTful interfaces WebSocket
data
Storage Controllers
Überdust
Input-Output Formatters

Fig. 1. High level architecture

of Überdust, is that it focuses not only on collecting data but also on allowing bidirectional communication between IoT applications and smart objects. Überdust offers a centralized control mechanism for all devices together with the storage of historical data that due to space limitations cannot be stored on the devices themselves. All messages to and from the devices pass through Überdust, which in general terms provides the functionality of an entry point so that any application can communicate with IoT devices and vice-versa.

On top of Überdust, the highest level of our system is the IoT application layer for monitoring sensor readings and controlling actuators. Most users do not actually need or have to interact directly with the sensor devices but with the abstraction provided at this level. This abstraction is one of the most interesting features of Überdust, as smart object devices can be grouped based on their semantic descriptions. We call this abstraction a Virtual Node, that is, a grouping of real devices as a common semantic entity. Usually in Multi-office or Smart building scenarios such semantic entities cover small areas like workspaces and offices, rooms, building floors or equipment groups like the heating or plumbing. We believe that Semantic entities (i.e., Virtual Nodes) are more suitable for user interfaces than the smart objects themselves as they represent in a more natural way the functions they support. Finally, this layer also includes a number of applications that are not designed for direct user interaction, like the applications that control lights based on the occupancy of the rooms or workplaces.

2.1 Integration of Applications

Überdust offers RESTful (HTTP and CoAP) and Websocket APIs for applications and smart objects. The RESTful API, based on the REpresentational State Transfer web service model, offers an easy-to-use, resource-oriented model to expose services. Publishing and consuming functionalities are implemented using both the HTTP and CoAP, Get and Put methods to represent data exchange. CoAP support is based on Californium [10], a library for parsing and generating CoAP messages, allowing direct top to bottom communication.

The WebSocket API offers the same functionalities using the IETF standardized WebSocket protocol [7]. WebSocket is a web technology providing bi-directional, full-duplex communications channels over a single TCP connection which can be used by any client or server application. Using a Websocket connection applications can receive new and historical readings or issue commands to actuators. Also, controller devices can use a Websocket connection to continuously stream new readings to the storage web service and receive actuator requests to forward to the sensors. All data messages exchanged using the WebSocket API are serialized using Google's Protocol buffers[5] a language and platform independent, extensible mechanism for encoding structured data in an efficient and extensible format.

2.2 User Interfaces

Überdust itself offers a very basic interface with no user authentication mechanism for publishing, accessing and sending information to the system as it is designed solely as a M2M system. So to provide all the above we designed applications on top of Überdust. A Drupal Website was designed to provide web based access to the system with specific drupal modules for operations, like switching lights on, controlling the HVAC units and accessing information in using time series diagrams, heat maps or pie charts. Also, as smartphones are always present in our everyday life, we designed two mobile applications for the Android and the Windows Phone platforms. The mobile phone applications offer the same operations as the Web interface, adapted to the special characteristics of each platform, focusing on the benefits of mobile devices. Both Drupal and Mobile applications use an authentication mechanism to address issues like privacy and security which are prerequisites in a Multi-office environment.

2.3 High Level Description of Available IoT Technologies

CoAP is another basic component of our architecture. All of the application and the smart objects we designed are CoAP enabled. This means that they exchange information in an standardized format, that all layers of our system can understand. The CoAP messages exchanged also contain semantic descriptions for observed events. Semantic descriptions offer access to some other IoT technologies like RDF [9] descriptions and SPARQL [13] queries. RDF is a standard model for data interchange on the Web that supports the evolution of schemas over time without requiring any changes to data consumers. RDF describes properties of devices with structured triples that can be exchanged between applications. SPARQL is a query language used to express queries over RDF structured data.

[5] `http://code.google.com/p/protobuf/`

SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. As a result all IoT applications can eventually be described in a completely abstract model as a number of SPARQL queries over the RDF data provided by the smart object network.

2.4 Automatic Configuration

All the applications and hardware platforms we designed to operate with minimal human intervention. Smart objects are automatically registered on the web application when they are turned on within the communication range of controller, while updates on their metadata (e.g., location) are available only by the administrator of the system. All interfaces or applications are configured based on the information of the Überdust web application. Users or administrators simply need to provide the `url` of the Überdust endpoint and then menus and information or control pages are generated based on the metadata and the semantic descriptions available. As a result installation of a similar system in a different environment or building is simplified.

2.5 Use Case

The major problems we target with our work are energy conservation and intelligent building configuration. Individual rooms, offices, or even entire buildings should go into energy saving mode when unoccupied disabling any communication networks, light, climate control, warm water or standby appliances to significantly reduce their power requirements. Implementing such behaviors in low level, independent systems with no access to information about the general situation in the building is extremely difficult and not at all adaptive to changes. IoT technologies on the other side provide all the necessary tools, described above, to deal with such problems with more adaptive and targeted solutions. Also by using external information about user preferences and requirements we can create a more personalized and pleasant experience.

3 Hardware

In order to integrate the main electric and electronic appliances with the Web we developed a series of hardware boards. Our main design goals is to support wireless communication, remote programming, be inexpensive, and well supported by the open source community to get feedback. We used an Arduino Pro Mini, combined with a IEEE 802.15.4 XBee® RF module.

3.1 Smart Lamps

The Arduino boards we designed are attached to ceiling and desk lamps in every room of P-Space. To control the $230V$ AC Lamps from the $5V$ I/O pins of the Arduino we use relays. Each board is able to control up to two different lamps so that we can provide users with multiple levels of lighting, e.g., low during a sunny day or high during the night. This allows us to reduce the power requirements as people tend to turn all the lights during the whole day. Furthermore we decided to introduce a manual override in case of hardware failure.

On top of the above, boards are also equipped with sensors for measuring luminosity, temperature and motion via Pyroelectric infrared sensors (Figure 2).

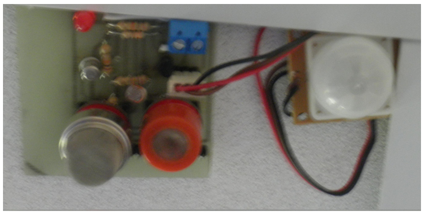


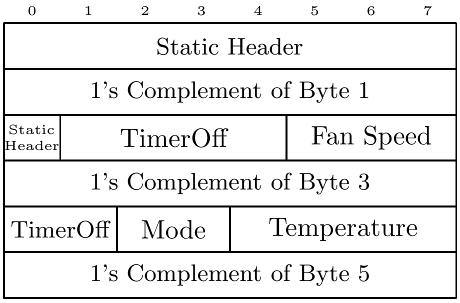
Fig. 2. Sensor Board for Smart Lamps

The Smart Lamps are operated by an application running at the top layer of our architecture. The sensed values for luminosity are periodically reported to Überdust and motion events generated by the PIR sensor are reported on a per-event basis. The application combines these values to decide when to turned on and off the lamps to provide better working conditions. To achieve the requirement of an energy-saving application when no people are present in the room or building (based on the readings provide by the PIR sensors), an energy saving scheme is initialized. At this point only lamps are turned off to conserve energy, but in the future the target is to extend the installation to control devices like printers or monitors that operate in stand-by modes while not in use but still conserve even small amounts on power.

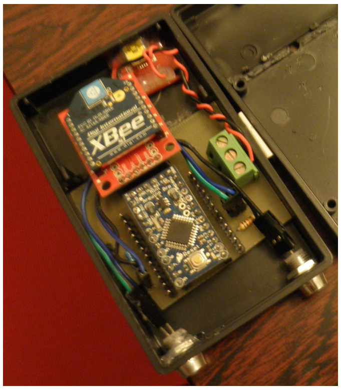
3.2 Smart HVAC

The HVAC controller is a developed in order to substitute the 38 KHz IR controllers used (Toyotomi brand) by an Arduino. For this purpose we developed a software library which allows us to autonomously control a wide range of HVAC devies, that are compatible with a specific IR protocol used widely in this particular area. The library, provides an API which substitutes all functions supported by a `R51L1/BGE` remote controller used principally by Toyotomi. Additionally, the API provides us an extra function in order to send a raw input signal, modulated in 38 KHz carrier frequency.

Our library is structured so that it supports scalability by adding new, more complex functions, or functions compatible with a wider range of HVAC models even modulated in different carrier frequencies. As an overview of the IR protocol, we see the following general characteristics in the majority of the supported functions:



(a) Toyotomi Control payload



(b) HVAC control board

Fig. 3. HVAC controller Payload and Arduino Board

1. The useful information consists of 6 bytes which are complementary in pairs at the level of bits.
2. In most functions, the three non-inverted bytes (1st, 3rd and 5th) contain coded the whole new state, represented in absolute values.
3. Every single bit of the above 48 ones, is encoded as follows: $1 \rightarrow 1000$ and $0 \rightarrow 10$.
4. A static head and tail piece of information is added to the above signal.
5. In the final input signal, every single bit take 21 cycles of 38 KHz frequency, i.e. about 553 μs.
6. The input signal is modulated with a 38 KHz carrier signal and the final one drives an IR LED in order to be transmitted to the HVAC unit.

Figure 3(a) shows how data are structured inside a control message and in Figure 3(b) depicts the final version of the control board in its casing.

3.3 Security Sensors

Another important part of the system is the sensors that report the state of the windows and doors (open/closed). Although the operation of these sensors is pretty simple and straightforward their operation is important in many other sub systems. To understand whether windows are open we used the very simple and widely tested Hall effect sensors used in home alarm systems. This obvious benefit from this installation is that the infrastructure can be used as a highly advanced low-cost alarm system, that can facilitate sophisticated notification systems. The list of supported notification targets include mobile phones via SMS, email, prerecorded telephone calls even using technologies like Skype, desktop notification systems in addition to the common but always effective sound and visual alarms. A second very important contribution of this application is that based on the state of the windows we can disable the operation of the HVAC units as it is completely environmentally unfriendly, e.g., to use Air conditioning while windows are open.

3.4 Smart Faucet

This board provides control of a water valve in an integrated method to the general system designed. It allows controlling the water supply to avoid flooding or detect possible leakages that may result is loss of water. The board designed operates in 3 levels. Locally it can control a vale in order to provide water by a faucet in the bathroom. Also the initial target was to use two valves in order to provide water in a predefined acceptable temperature but this proved to be unrealistic due the operation restrictions of the valves, as it was not possible to control the flow of the water accurately. A second part of the board detects flooding with a simple circuit that short-circuits submerged in water. If this is detected then the water supply is stopped to prevent problems with electrical equipment and possible damage to furniture or the building.

A `Sharp GP2Y0A02` infrared distance sensor was used to detect hands placed under the faucet. The sensor measures the distance of any obstacle within a range of 5-120cm. Distance is measured via the voltage change on the two ends of the sensor as a resistor using the analog inputs of the Arduino. Voltage level is polled using a time interval of some milliseconds and the measurements are translated to distance values measured in cm.

Calculating the exact distance was not as important as detecting the any obstructions (e.g., hands) under the faucet is the only case we examine, thus the sensor's precision was inadequate. When the distance measured by the sensor, drops below 35cm, which is the distance between the sensor and the washbasin, the water valve is activated, letting the user use the faucet.

4 CoAP

The Constrained Application Protocol (CoAP), is a draft by IETF CoRE working group which deals with Constrained Restful Environments. It provides exactly the subset of HTTP methods (`GET, PUT, POST, DELETE`) which is necessary to offer RESTfull web services in WSNs. We work with the implementation of the 8th version [12] of CoAP presented in [4]. Messages follow a specific message format, which is simple enough to be processed and used by both IoT desktop applications and smart objects.

4.1 Quality of Service

CoAP communication between endpoints is based on a lightweight request/response model. Message exchange is asynchronous and based on UDP as reliable and unreliable CoAP. With the unreliable model, endpoints transmit their messages and there is no way to confirm delivery of each individual message. On the reliable model, acknowledgement messages are sent upon delivery, something similar to TCP, acknowledging the other endpoint for the receipt of the message. Messages that are not acknowledged, either because of communication or hardware failure are retransmitted up to 4 times in exponentially increased time intervals. Responses are by default piggybacked on the acknowledgement messages so that the actual response is part of the acknowledgement. Piggybacking is also enabled by default to reduce the messages exchanges in half and thus reduce the traffic inside the network and the total power requirements. Separate response messages if the applications request it explicitly.

4.2 Notification Mechanism

A common problem when dealing with active sensors is how often we need to request an updated value for the sensor. We always want to know the latest value available, but working on WSNs with many constrained devices, would be catastrophic for the network's efficiency and the power consumption to request new values about multiple resources every a few seconds. To solve this problem CoAP introduces the **Observe** [8] CoAP extension which defines a mechanism for clients to register as observers and for servers to push updated resource representations to interested clients, while still keeping the properties of the RESTful interface. New values are automatically pushed to the registered clients while clients can define threshold values in order to be notified only when the updated values fit their interests. In order to get a fresh value even if it's below the threshold, CoAP Observe, periodically pushes the latest values to clients, like a notification that the device is still functional.

4.3 Discovery of Resources

The discovery of resources offered by a CoAP endpoint is extremely important in machine-to-machine applications where no humans intervene in the loop and static interfaces result in fragility. Our CoAP endpoints support the CoRE Link Format [11] and offer the `.well-known/core` resource which responds with all the available resources on the server. This defines how a CoAP endpoint can inform a Client of its resources, in a format that is recognized from both ends. Together with the resource `URIs` several attributes can be included (like Semantic descriptions), offering information about the resource type, the interface description, the expected size or even a text description. Explicit request of available resources from one CoAP endpoint is avoided and mainly done during the auto configuration phase. Client requests for available resources and nodes, are directly answered by the gateway where all nodes register themselves.

4.4 Sensing and Actuation with CoAP

As mentioned before, we use CoAP to facilitate communication between IoT applications and smart objects. CoAP offers this functionality through the RESTful api available on the Endpoints. Like in the HTTP protocol, CoAP offers

GET,PUT,POST & DELETE methods for interaction with the resources of the end-point. In our system we use only GET and PUT. GET resources describe sensing capabilities. Applications can register on them to receive notification for all the new values that become available. PUT resources, on the other hand are used to describe the actuator of our system. When actuators receive a put request for one of their resources the equivalent action is performed.

5 Drupal Web Interface

User interfaces are a key factor for every system. In order to make the automations of P-Space accessible to end users, we needed a friendly, easy-to-use environment accessible from everyone everywhere. We developed a web site using the open source Drupal CMS, because of it's flexibility, reliability and security.

Using Drupal's API combined with our technologies, we developed a package of modules to view the status and control the actuators of P-Space. The communication with Überdust is done using the RESTful interface and JSON formatted data. Sending commands is limited to certain authorized users only, utilizing the easily configurable permission system of Drupal. The modules developed are :

Monitor Module: Users and guests can view the location and status of every node in P-Space, over a bird's eye view of the building. To print the image, we used the Scalable Vector Graphics (SVG) format, which can be produced dynamically from our code and create areas for each node which can then be easily handled with JavaScript to produce effects on specific events or to change styles based on the state reported by the sensors.

Control Interface: Users can view the status and control lights in each room by clicking on switches. On click, via Javascript an HTTP request is sent to Überdust using the RESTful interface in order to trigger the suitable action on the smart lights hardware. Instead of continuously polling the status of all the lights in the building we use WebSockets to receive notifications of the changes in the state of the smart lights in real time. Polling is a feature that has adverse effects on both the utilization of the available bandwidth as well as on the operation of the core web service.

HVAC Control: A user interface created on top of the actual image of the remote control of the HVAC units is used to control it via the website. Settings for the two fully controlled units in the two rooms change using Drupal's Form API and then the commands are sent once again to Überdust using the RESTful interface.

Finally, each smart object is available on its own page, where we can retrieve all the associated information. This interface supports a special operation used mainly for debugging in the first stages of the installation of the system where users can send explicit binary messages to the smart object.

All modules can be easily configured and extended to include specific support for any new devices added to Überdust. All information for the devices like the position and the capabilities is actually stored using Überdust and configuration done via request to the RESTful interface resulting to a completely adaptable system to all changes that may happen like device failures, repositioning of devices or extension of the underlying network.

6 Mobile Phone Applications

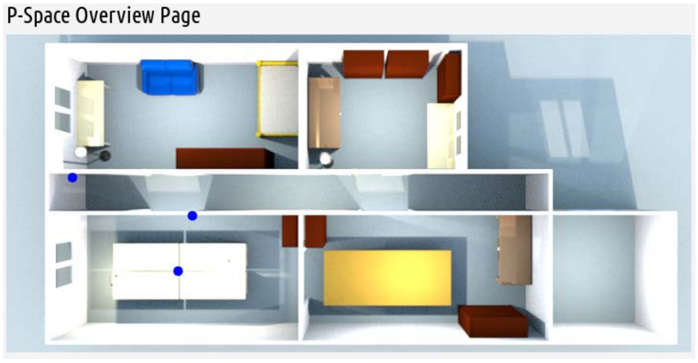
We develop an alternative approach to monitoring and controling P-Space that is available for Android®and Windows Phone®. Both implementations act as clients for Überdust web services and offer browsing of sensor readings and actuator status based on the privileges of each user. Some of the features offered by the Überdust `mobile applications` are:

- the authentication mechanism where, after launching the application, users needs to enter their credentials to get access to the functionality provided by their role.
- Important information about a specific workplace or room, like temperature, humidity, luminosity, the status of a lights, or the current consumption of electrical devices (computers, a/c units), etc.
- A notification mechanism for special aspects like security security through the notifications from motion sensors or sensors on windows and doors.
- Also we can simplify our life by having remote access to control electrical devices such as boilers or a/c units without the need to search for the remotes, or switches.

The Android application was developed for Android 2.2, API Level 8. Connection with the web server is accomplished through a RESTful interface by using JSON formatted data. The application needs a minimal bandwidth to operate on permanent basis as data for sensors are cached and not retrieved every time the user launches the application. Also, the user interface uses the external android library `ViewPagerIndicator` which allows us to have smooth transitions by sliding through different views. The `GraphView` library also added support for graph of the historical data available on Überdust to have better view of how sensor readings change over time in order to identify special events (e.g., power spikes or sudden temperature changes).

The Windows Phone application was developed for Windows Phone 7.1 using "Microsoft®Visual Studio Express for Windows Phone" SDK. Its functionality is similar to the Android application and operates using Überdust RESTful interface to retrieve data in JSON format.

Both applications offer support for multiple profiles where users can setup different Überdust servers to use (e.g., their home and office buildings). Retrieved information is sorted based on the location of the devices. Users can also go through a different list based on the functionality each device or room offers. Especially for actuators authenticated users can send commands to nodes that are controlling devices like setting the temperature of the a/c unit or switch on lights or the boiler.



7 Conclusion

The Internet of Things is a technological revolution that represents the future of computing and communications while its development depends on dynamic technical innovation in a number of important fields, and especially in wireless sensor networks. Connecting sensory networks to the Internet creates endless opportunities for applications and services, new emerging models of operation.

Currently many researchers are working on designing, developing and evaluating new protocols, embedded IP stacks and operating systems for WSNs. Many ongoing projects are aiming at interconnecting WSNs with the Internet and establishing programming environments for developing applications.

In this work we presented our efforts in designing smart services for a multi-office building, developing and deploying hardware extensions for electric and electronic appliances and integrating their operation with the Web. Our work shows how IoT devices can be applied to real world scenarios and installations by using very recent technologies and open standards.

Indeed some methodologies and technologies such as CoAP, RESTful interfaces, WebSockets, JSON format are very usuful fo the integration of the software systems across the IP-stack with the hardware devices. Furthermore the Virtual Nodes abstraction offered by Überdust was very helpful in reducing the complexity of controlling multiple devices as a single logical device and aggregating readings from multiple sensors and across different time windows. We reported newly-developed Drupal modules that simply implement application logic and provide human-computer-interaction using Web technologies. Similarly the smartphone applications exploit the same RESTful interfaces and WebSocket APIs in order to access the current state of the devices in JSON format. Both environments offer all the necessary tools to establish communication and manipulate the data received with minimum programming effort.

However, we believe that our approach was complex and required a big team for the development of applications exploiting the merged infrastructure. The current methodologies & technologies available are not enough to have a tremendous impact on the development of Future Internet applications. This is due to the fact that despite the IP-based integration of the embedded world, application-level protocols, software and development environments, but also design and evaluation methodologies in the Internet and in the embedded world are vastly different and lack integration. An application developer currently still has to bridge this gap manually; he has to be an expert in both worlds.

There is a lot of future work that needs to be done in this direction so that devices can be self-configured using semantic descriptions and by minimizing human intervention. It is important to work towards infrastructures that can adapt to dynamic changes in the environment as new devices are introduced, rellocated or removed from the space.

Acknowledgements. This work is partially supported by the European Union under contract numbers ICT-258885 SPITFIRE.

References

1. Kansal, J.L.A., Nath, S., Zhao, F.: Senseweb: An infrastructure for shared sensing. In: IEEE MultiMedia, pp. 8–13 (2007)
2. Aberer, K., Hauswirth, M., Salehi, A.: The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks. In: 7th Int. Middleware Conference (2006)
3. Akribopoulos, O., Amaxilatis, D., Chatzigiannakis, I.: Towards integrating iot devices with the web. In: 7th IEEE Conference on Emerging Technologies & Factory Automation, ETFA 2012 (2012)
4. Amaxilatis, D., Georgitzikis, V., Giannakopoulos, D., Chatzigiannakis, I.: Employing internet of things technologies for building automation. In: Conf. on Emerging Technologies & Factory Automation, ETFA 2012 (2012)
5. Corcho, O., García-Castro, R.: Five challenges for the semantic sensor web. Semantic Web 1(1,2), 121–125 (2010)
6. Dawson-Haggerty, S., Jiang, X., Tolle, G., Ortiz, J., Culler, D.: Smap: a simple measurement and actuation profile for physical information. In: 8th ACM Conf. on Embedded Networked Sensor Systems, SenSys 2010, pp. 197–210 (2010)
7. Fette, I., Melnikov, A.: The websocket protocol. Proposed statndard, IETF (2011)
8. Hartke, K.: Observing Resources in CoAP. Internet-Draft, IETF (2012) (work in progress)
9. Hayes, P. (ed.): RDF Semantics. W3C Recommendation. World Wide Web Consortium (2004)
10. Kovatsch, M., Mayer, S., Ostermaier, B.: Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things. In: 6th Int. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2012 (2012)
11. Shelby, Z.: CoRE Link Format. Internet-Draft, IETF (2011) (work in progress)
12. Shelby, Z., Hartke, K., Bormann, C., Frank, B.: Constrained Application Protocol (CoAP). Internet-Draft, IETF (2011) (work in progress)
13. SPARQL query language for RDF. Technical report, World Wide Web Consortium (2008)