Towards integrating IoT devices with the Web

Akribopoulos Orestis, Amaxilatis Dimitrios and Chatzigiannakis Ioannis
Computer Technology Institute & Press (CTI), Patras, Greece
Computer Engineering and Informatics Department, University of Patras, Greece
{akribopo, amaxilat, ichatz}@cti.gr

Abstract

In this paper, we discuss the integration of Wireless Sensor Networks (WSN) and smart objects with the Web. We present a set of research challenges which we believe are the most important ones rising from this integration and propose a prototype system, Überdust, which addresses such challenges. Überdust is 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. Our system provides high-level language-independent APIs so IoT application developers may choose their favorite programming or scripting languages.

1. Introduction

The Internet of Things (IoT) refers to the integration of uniquely identifiable Smart Objects, embedded devices of real-world semantic entities and their virtual representations to the Internet. IoT application developers operate on a level above the hardware WSNs, but the methodologies for deploying and annotating these sensor devices as smart objects are neither straightforward nor standardized making application development for the IoT tied up to the characteristics of WSNs. IoT systems need to address fundamental problems [7] like :

- hardware, software and networking heterogeneity,
- intermittent connectivity,
- application scaling issues,
- simplification of the development and deployment cycle,
- no standardized service and capability descriptions,
- big data management

There has been a wealth of activities in the context of semantic description of WSNs [5, 6] and IoT that resulted to a number of experimental and production oriented systems focused on collecting data from a variety of sources.

GSN[1] [2] (Global Sensor Network) is a software middleware designed to facilitate the deployment and programming of sensor networks consisting of over 80 million sensing data points.

Cosm[2] (previously known as pachube) is a secure, scalable platform that connects devices and products with applications to provide real-time control and data storage. Cosm provides an online database service which allows developers to build their own applications based on the Cosm sensor-derived data.

sMap [8] (Simple Measuring and Actuation Profile) is a RESTful web service which allows instruments and other producers of physical information to directly publish their data. Its strengths are that it is easy to consume, easy to implement for new device types, and simple to process.

SenseWeb[3] [1] is a prototype system which provides a `.NET` API. Applications using SenseWeb can initiate and access sensor data streams from shared sensors across the entire Internet. SenseWeb allows multiple applications to share concurrent common sensing resources.

In this work, we present a new system, **Überdust**, which addresses the above issues and provides storage, sharing and discovery of real-time and historical data from smart objects, WSNs and building installations around the world via the Web. Überdust is a data storage and brokerage web service to enable 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 of Überdust, is that it focuses not only on collecting data but also on allowing bidirectional communication between IoT applications and smart objects. Überdust acts as an entry point to the smart objects network, providing functionalities for interaction with one of the smart objects registered.

2. Überdust

Überdust was developed around an ontology that represents the common set of terms defined by the W3C Semantic Sensor Network Incubator Group [12] for sensors, sensor networks and sensor web applications, along with

[1]http://sourceforge.net/apps/trac/gsn/

[2]https://cosm.com/

[3]http://research.microsoft.com/en-us/projects/senseweb/

their properties and events. Smart objects and Sensor Networks are similar to each other, thus the research results on WSNs can be applied to solve an important subset of the IoT issues. We use the Semantic Sensor Network ontology, as it provides access to important data and metadata that facilitate abstraction, categorization, and reasoning offered by semantic technologies like RDF [10] and SPARQL [13]. The extracted entities that compose the Überdust ontology are:

- *Testbed*, a representation of user defined groups of smart object networks or WSNs installations.
- *Node*, as a representation of smart objects (Sensors & Actuators).
- *Link*, as a representation of relations between Nodes.
- *Capability*, as a property of Nodes or Links like position, sensing abilities or a description.
- *Node & Link Readings*, as the values that describe a Capability over time.

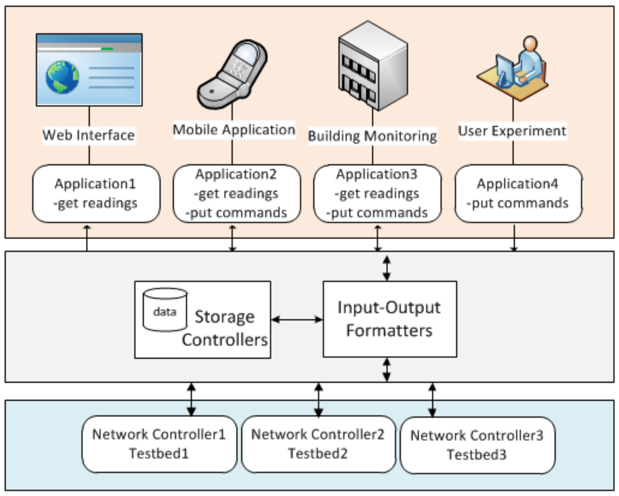
Web Interface
Mobile Application
Building Monitoring
User Experiment
Application1 -get readings
Application2 -get readings -put commands
Application3 -get readings -put commands
Application4 -put commands
data
Storage Controllers
Input-Output Formatters
Network Controller1 Testbed1
Network Controller2 Testbed2
Network Controller3 Testbed3

Figure 1. Überdust Architecture

Überdust follows the 3-tier model presented in Fig. 1, where the storage service interconnects the low level, smart object networks with the top level Applications. The main concept of Überdust architecture is that all layers share a common ontology model.

On the lower level there is one or multiple groups of sensor devices, attached to or monitoring real world objects. Überdust was initially designed to monitor and control WSNs but during the implementation of the system became obvious that any monitored device can be part of a Testbed as long as it either provides sensor readings (e.g., CPU temperature) or can be controlled as an actuator (e.g., an existing proprietary Building Management System).

The top level of Überdust is the IoT Application level. Sensor readings from the testbeds are consumed by applications to either produce a simple chart of the temperature from a single sensor or to control a whole building (e.g., an Intelligent Building Monitoring System).

The middle level is a data storage and brokerage web service responsible for providing real time and historical data of the sensors through the implemented APIs in various formats, connecting the Sensor networks and IoT applications. All incoming data from the smart objects are inserted to an internal storage mechanism. Applications are notified of the new values instantly while commands are routed to the respective smart object actuators.

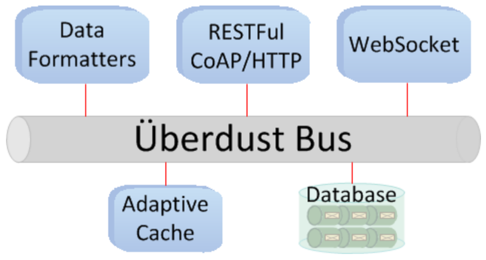

Figure 2. Central Überdust ESB Service

3. Implementation Details

The core components of Überdust integrate using an Enterprise Service Bus (ESB) architecture, which is a software architecture model for distributed computing with asynchronous message oriented communication. All modules of Überdust communicate over the Überdust Bus, as shown in Figure 2. ESB requires all modules to have generic interfaces that hide internal implementation choices while it increases the systems scalability, modularity and openness to new extensions. ESB also brings communication between different modules down to the exchange of basic common messages thus providing an open and agile platform for expansion and integration with other applications available from the open source community.

3.1. Communication API

Communication between the 3 layers of Überdust is achieved using the Überdust RESTfull and Websocket APIs. Users and sensor network administrators can use the API that suits their needs better and communicate with the Storage Service. All requests, from applications and Network Controllers are inserted to the internal Überdust storage and forwarded to the registered clients.

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 Constraint Application Protocol (CoAP) `Get` and `Put` methods to represent data exchange. CoAP support was added on Überdust, using Californium [11], a library for parsing and generating CoAP messages, allowing direct end-to-end communication. As CoAP is not yet implemented in many hardware platforms we implemented a CoAP library for SunSPOT and Arduino sensor devices as well as on the Wiselib [4] Algorithms Library that supports among others iSense, Contiki and tinyOS devices.

The WebSocket API offers the same functionalities using the IETF standardized WebSocket protocol [9]. 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, Network Controllers 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[4].

3.2. Generic Database

The base of our system lies in using a database for storing the information of the Überdust ontology without strict definition of the actual storage system. We use JPA and Hibernate Annotations to describe the objects and relational mapping of our model as Java Persistent objects and implement Storage Controllers to execute queries on the actual database used (MySql, Hadoop etc.). Controllers provide all possible operations from Insert and Delete statements to complex conditional queries like aggregates, joins or initialization with a single function call.

The base element of the Überdust ontology is the **Node**, representing any device (sensor or actuator) of the network. **Capabilities** of Nodes and Links cover a wide range of attributes that either describe or characterize smart objects in the context of IoT and are actually divided into 3 categories: 1) **sensing** capabilities (i.e., temperature, humidity) representing new readings from smart objects, 2) **actuation** capabilities (i.e., switches, controls) that describe actions performed by the actuators inside the network, 3) **metadata** capabilities like the position of the node or its description usually set by the administrators of the system.

Formatters: Formatters translate Überdust ontology objects to and from a wide range of formats in computer and WSN applications. They are designed to operate both as part of the web application and as a standalone module that given an input of Readings presents them in a number of well known and widely used formats like HTML, JSON, XML, CSV, KML, WiseML and SensorML. Formatters are used in many parts of Überdust to facilitate communication between different modules.

Moreover, the adoption of the ESB architecture allows us to better design and implement a multiple level caching schema. Based on the above architecture, we have identified that the performance of every component of our system can be improved by caching. Thus, the messages that are exchanged inside the Überdust Bus can also pass through a caching mechanism that associates each message with a time-to-live (TTL) value to differentiate caching strategies for different types of messages. Another feature of the above caching schema is that it provides an adaptive caching scheme. This is accomplished by automatically selecting the most suitable caching policy based on the changes in the workload, the network load and the total number of concurrent requests.

[4]http://code.google.com/p/protobuf/

3.3. Network Controller and Sensor Applications

The lower level of Überdust, the Network Controllers are responsible for dealing directly with the smart objects and their unique characteristics. As most WSNs deployed at the moment are neither IPv6 nor CoAP-enabled, this layer is responsible for connecting the devices to the web application and hides out all deployment-specific characteristics, providing an abstraction of a stream of messages from the WSN to Überdust and back. Network Controllers essentially act as translators for application and sensor level communication. In CoAP and IPv6-enabled WSNs the operation of Network Controller is more straightforward, as the only translation is between 6LowPan to IPv6 packets, as CoAP messages are common and can be processed by all 3 levels of Überdust. On the other hand, dealing with a plain 802.15.4 or RF network is harder, and a specific communication layer has to be implemented by the administrator of the network. For 802.15.4 networks we implemented the XBee Network Controller that uses an XBee to communicate with the WSN and WebSockets to communicate with Überdust. We implemented a communication protocol, which is compatible with many sensor devices (Arduino-XBee, iSense, TelosB and SunSPOT devices were tested) using the mkSense [3] library.

mkSense allows bi-directional communication between hardware platforms that use 802.15.4 compliant radios but due to partial implementation of the standard, do not communicate out of the box. The exact format of the messages exchanged is depicted in Figure 3.

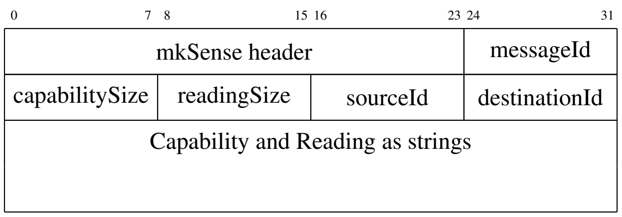

Figure 3. 802.15.4 **Report & Control protocol**

The header contains the mkSense header required for heterogeneous communication and the `messageId` required to differentiate reporting and control messages. For Link Readings, the header contains 2 Mac addresses of the two ends of the Link. For Node Readings, the second address is equal to the broadcast address (`0xffff`) to differentiate from Link Readings. Additionally, two size counters for the strings representing the Capability and Reading are part of the header. The payload of the message contains the two strings describing the capability and the reading reported. Control messages follow the same format with a different `messageId`.

3.4. Top Level IoT Applications

Überdust was designed to facilitate the development of applications that use real time and historical data from smart objects and WSNs. The RESTful and Websocket APIs were adopted to that direction as both are well supported across many programming and scripting languages like Java, Python, Javascript or even Unix Bash shell.

Virtual Nodes: The concept of Virtual Nodes addresses the problem of grouping hardware devices to represent Real World Entities that cover multiple hardware devices like Rooms or Buildings. Virtual Nodes are a combined representation of all the Capabilities of nodes that compose them. Readings for the sensing capabilities of the virtual nodes, aggregate real Node readings while actuator actions are forwarded to all grouped smart objects. The Virtual Nodes are stored in Überdust as the hardware devices, with the prefix "virtual" in their name. Virtual Nodes are associated with the devices that combine them in the database with links of type "virtual". This form of "defining" virtual nodes allows to easily extend on multiple levels and have Virtual Nodes that consist of other Virtual Nodes.

4. Discussion - Future Work

In this paper, we discussed the integration of WSNs and smart objects with the Web, along with a set of research challenges which we believe are the most important ones rising from this integration. Überdust is a prototype system, that tries to answer such challenges, and we briefly presented its architecture and implementation.

Our future work includes the implementation of additional features, the refinement of our implementation, especially with respect to the system architecture. An interesting additional feature is a regular expressions mechanism so that users can create more powerful implementations. As objects of the WSN report new measurements constantly, it is possible to receive same values for a long period of time as new readings. Using Regular Expressions users are able to define the data they want to receive notifications for. Based on the functionality provided by the Regular Expression mechanism, we also introduce another feature we call Virtual Capabilities. Virtual Capabilities are defined by a regular expression based on one or multiple node or link capabilities of real or virtual nodes. For example, a user may indicate that the virtual capability room_in_use is true when the pir sensors report movement or the noise level high, and will thus be notified only when the above conditions are satisfied.

Finally, we would like to extend our system with authentication, authorization and accounting (AAA) functionality. Authentication, for confirming that a user or an application has the required permission to interact with the system, authorization for specifying access rights to specific resources and accounting for tracking of network resource consumption by users (e.g., for billing purposes).

We have already started using Überdust in various deployments in multi-office environments. We expect to receive valuable feedbacks and also carry out an extensive performance evaluation.

Acknowledgements

This work is partially supported by the European Union under contract numbers ICT-258885 (SPITFIRE[5]).

References

[1] J. L. A. Kansal, S. Nath and F. Zhao. Senseweb: An infrastructure for shared sensing. IEEE MultiMedia, pages 8–13, 2007.

[2] K. Aberer, M. Hauswirth, and A. Salehi. The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks. In *7th Int. Middleware Conference*, 2006.

[3] O. Akribopoulos, V. Georgitzikis, A. Protopapa, and I. Chatzigiannakis. Building a platform-agnostic wireless network of interconnected smart objects. In *Panhellenic Conf. on Informatics*, PCI '11.

[4] T. Baumgartner, I. Chatzigiannakis, S. Fekete, C. Koninis, A. Kröller, and A. Pyrgelis. Wiselib: A generic algorithm library for heterogeneous sensor networks. In *7th European Conf. on Wireless Sensor Networks*, EWSN '10, pages 162–177.

[5] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.

[6] M. Compton, C. Henson, H. Neuhaus, L. Lefort, and A. Sheth. A survey of the semantic specification of sensors. In *2nd Int. Workshop on Semantic Sensor Networks*.

[7] O. Corcho and R. García-Castro. Five challenges for the semantic sensor web. *Semantic Web*, 1(1,2):121–125, 2010.

[8] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. smap: a simple measurement and actuation profile for physical information. In *8th ACM Conf. on Embedded Networked Sensor Systems*, SenSys '10, pages 197–210.

[9] I. Fette and A. Melnikov. The websocket protocol. Proposed statndard, IETF, 2011.

[10] P. Hayes, editor. *RDF Semantics*. W3C Recommendation. World Wide Web Consortium, 2004.

[11] M. Kovatsch, S. Mayer, and B. Ostermaier. 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 '12.

[12] L. Lefort, C. Henson, K. Taylor, P. Barnaghi, M. Compton, O. Corcho, R. García Castro, J. Graybeal, A. Herzog, K. Janowicz, H. Neuhaus, A. Nikolov, and K. Page. Semantic sensor network. In L. Lefort, C. Henson, and K. Taylor, editors, *W3C Incubator Group*, number June, page 80. W3C, 2011.

[13] SPARQL query language for RDF. Technical report, World Wide Web Consortium, 2008.

[5]http://spitfire-project.eu