Assignment
Master of Applied Computer Science

Student name	Martin Lehmann
Student number	700766
Signature	Martin Lehmann

The signature of the student testifies that all content is the student's own work and that all sources are referred to.

Course code and name	MA220-14 Architecting the Internet of Things
Title	Data Analysis as a Service: an infrastructure for storing and analysing the Internet of Things
Hand out date	2015-03-10
Due date for submission	2015-04-10
Number of pages	8
Number of words	2,563

Data Analysis as a Service: an infrastructure for storing and analysing the Internet of Things

Martin Lehmann

3rd June 2015

Abstract

The Internet of Things (IoT) is perhaps the fastest emerging new technology trend, and must be deeply addressed through research. While we have many small-scale, single-component solutions for connecting parts of the Internet of Things, we have seen very few research oriented full-stack implementations for gathering, storing, and analysing data. We present Data Analysis as a Service (DAaaS), a full-stack sample implementation of a complete IoT application that accepts, stores, and provides both real-time and static access to the data. The main considerations of DAaaS include storage of time-series data in a regular document database, and machine-to-machine communication through standardised APIs. One common research challenge in the Internet of Things, *security*, is considered only briefly, and is of utmost importance in future research.

1 Introduction and background

Starting with Weiser's 'The Computer for the 21st Century' in 1991, we have seen an enormous development in computer science towards ubiquitous computing and interconnected physical devices – things.

The Internet of Things (IoT) is perhaps the fastest emerging technology trend of the present time. The IoT technologies and applications are still in their infancy (Xu, He & Li, 2014), and so the academic community must thoroughly address the area. Although 'IoT' was initially meant to describe a network of Radio-Frequency ID-enabled devices, it has since been expanded to the following widely accepted (Xu et al., 2014) definition:

> a dynamic global network infrastructure with selfconfiguring capabilities based on standard and interoperable communication protocols where physical and virtual Things have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network (Kranenburg, 2007 cited in Xu et al., 2014, p. 1).

It becomes clear that the Internet of Things indeed encompasses all devices with a sensor, but there is also a second implication: the humongous number of data points that will inevitably be collected is of no use to anyone unless it is processed. The definition also presents us with several implicit challenges, backed by Xu et al. (2014) and Palattella et al. (2013, 3). These include, but are not limited to, privacy, distribution and maintenance, and security concerns in the distributed system that is the IoT. These are all important areas to explore, but outside the scope of this article.

Also important to mention is the Web of Things (WoT): the software layer on top of the Internet of Things. This article mainly focuses on the programming model side of an IoT application, and is thus mostly concerned with the WoT.

Furthermore, *standards* are a real concern. This is described in Palattella et al. (2013, 3), which emphasises emerging industry alliances and IEEE/IETF working groups as the key to success.

Finally, the pre-eminent concern of this article is the gap of knowledge with regard to modelling and implementing complete IoT-oriented applications, as described by Paganelli, Turchi and Giuli (2014, 99).

This article first revisits the current state of research on the fields of the Internet and Web of Things, respectively. It then presents an architectural model and proof-of-concept implementation of a full-stack IoT-oriented application which accepts, stores, and provides access to the data in addition to subscription to real-time feeds for new data points. Third, it compares the experiences from modelling and developing the application to the existing research. Lastly, the most important lessons are highlighted and briefly discussed.

2 Literature Review: State of the Art

Xu et al. (2014) contributes a major review of the current research on the Internet of Things (IoT). This very recent (2014) survey paper with more than than 80 trustworthy references identifies several key gaps in the current knowledge body regarding the Internet of Things. The main points - cost, security, standardisation, and technology – are all areas that will need to be explored further, but only standardisation and technology are considered in this article. Additionally, they propose a service-oriented architecture (SOA) style approach to the Web of Things. This approach is not considered by this article.

As mentioned in the introduction, Paganelli et al. (2014, 99) describes a lack of actually modelled and implemented applications as a key hole in the current research body. This article also refers to a relatively large number of other articles proposing middleware and frameworks for designing applications in the Web of Things.

However, Palattella et al. (2013, 3) claims that what may have previously seemed impossible given the restrictions of the Internet of Things in terms of building a standards-compliant stack may indeed become a reality. They propose a highly technical communication stack for an entire application, but have not consider actually implementing a system. It is worth noting that their stack includes IETF's RFC 7252 - the Constrained Application Protocol (COAP) (2014) for application layer communication.

Xu et al. (2014) also mention context awareness as an important factor in the Internet of Things, as millions and billions of sensors will be connected, collectively producing extreme amounts of data. While not considered by this article, using context awareness and artificial intelligence to filter out meaningful, important data will be a great tool as we begin to find more and more use cases for the Internet and Web of Things.

It seems that there is no lack of proposed frameworks, protocols, and standards for connecting things to the internet and making them part of the web. There is no shortage of frameworks for the actual communication between devices and servers, either, and we have quite a few contributions regarding storage of very large numbers of data points. We also have much research on analysing the data on the field of Big Data, but that is outside the scope of this article).

Disregarding cost, privacy, and security, the main problem of the current Web of Things research body seems to arise only when committing to building a complete full-stack application: there is no standard, proven, manufacturer-independent way to implement a complete application for gathering and analysing data from a custom Internet of Things system. Indeed, as Xu et al. (2014) put it: the Internet of Things is still in its infancy.

3 The artefact: Data Analysis as a Service (DAaaS)

This section describes the design and implementation of Data Analysis as a Service (DAaaS), an artefact developed by Andreas Birn-Hansen and Martin Lehmann as a prototype project prior to the writing of this article.

A clear gap identified in the previous section is the lack of of sample implementations of full-stack applications where communication, storage, analysis opportunities, and availability are all thoroughly discussed and actually implemented. DSaaS is an attempt to start bridging this gap, but will naturally only provide the perspective of one domain, one technology stack, and one use case.

Very briefly, DSaaS accepts and stores data from providers (sensors), pushes the new data to a very simple customisable dashboard (see 3, and provides (optionally real-time) access to the data sets. Security is not considered in the prototype. It was implemented with the sole goal of building a complete application designed to handle data from the Internet and Web of things.

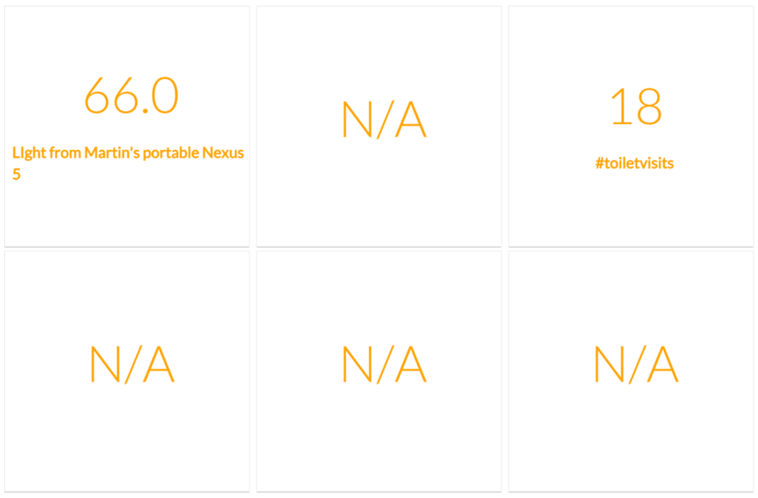
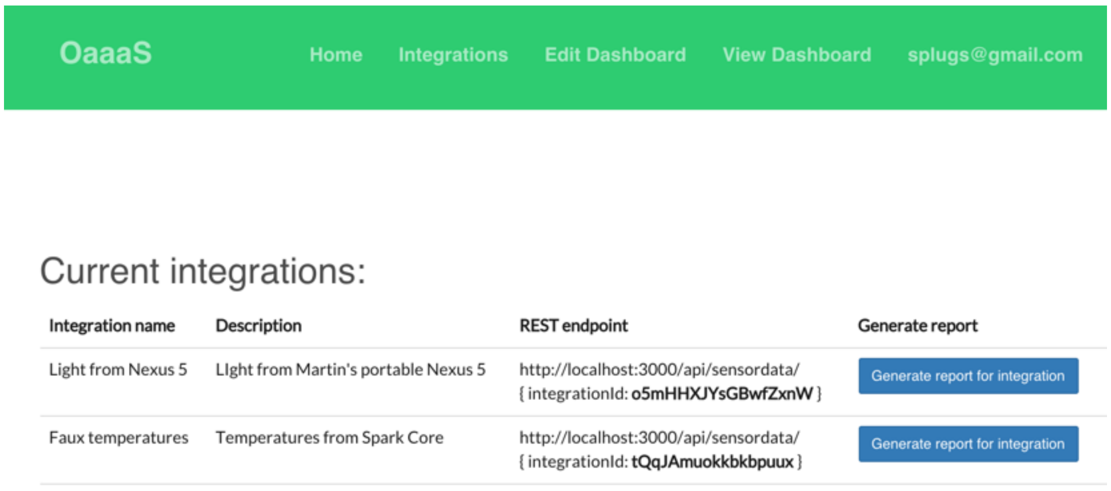


Figure 1: The simple dashboard with two integrations

The DSaaS core is a central server written in Meteor[1] providing access to both storing and retrieving data. It also provides the option of subscribing to a changefeed for a specific resource to receive updates to the dataset in real-time. DSaaS also provides a very simple real-time dashboard for monitoring incoming data. Finally, it provides a management interface for customising the dashboard and defining the *integrations* that can be displayed in the dashboard.



Current integrations:

Integration name	Description	REST endpoint	Generate report
Light from Nexus 5	LIght from Martin's portable Nexus 5	http://localhost:3000/api/sensordata/ { integrationId: **o5mHHXJYsGBwfZxnW** }	Generate report for integration
Faux temperatures	Temperatures from Spark Core	http://localhost:3000/api/sensordata/ { integrationId: **tQqJAmuokkbkbpuux** }	Generate report for integration

Figure 2: Sample integrations

An *integration* is a data provider of any kind that will upload data to the service. An integration is expected to be a single sensor whose data is sent to the internet - typically via an internet-enabled microcontroller – although it is possible to get creative. As seen in 3, creating an integration automatically generates a unique

[1]https://www.meteor.com/

ID, which must be included in requests to upload data as identification.

In addition to endpoints for storing data, DSaaS provides two different types of endpoints for accessing the stored data. The simplest of these is a traditional REST endpoint that exposes data from each sensor as a resource with a unique URI: an HTTP GET request fetches data from the present day. Of course, applying filters to fetch for example all stored data, data from the present week, or data from the last ten days, would be helpful – but this was outside the scope of the prototype.

The second data access endpoint provides a real-time changefeed that sends all new relevant data points to the consumer as it is stored in the database. The protocol for real-time data updates is Meteor's Distributed Data Protocol (DDP), which is based on WebSockets. It would naturally be possible to define a custom protocol with plain WebSockets. This enables building real-time graphs or custom dashboards for the data, or real-time analysis with for instance Apache Storm[2].

The prototype also includes three sample integrations/data providers (a Spark Core microcontroller[3], a native Android application listening for light values in the room using the light sensor on the mobile device, and a simple Ionic[4] cross-platform application for mobile and web for registering a single value.

Finally, the prototype includes one external real-time consumer written in JavaScript, which is a proof-of-concept realtime graph for a single sensor.

4 Discussion: experiences from developing Data Analysis as a Service

Unsurprisingly, many design decisions had to be made as we applied the Internet and Web of Things to a real-world application with a clearly defined use case such as Data Analysis as a Service (DAaaS). While frameworks for connecting things to the internet, machine to machine communication, data storage, and data analysis as plentiful, it proved impossible to apply these frameworks and protocol stacks to the application without modifications. In short, the development time can be greatly reduced by utilising tools which almost fit the use case, and customise what already exists. This experience differs from the main proposition in Palattella et al. (2013, 3), whose introduced IoT protocol stack should have been the best fit.

A key experience from the development process is that development time can be greatly reduced by using tools that already exists – in DAaaS's case, Meteor with MongoDB for storage, and REST and DDP[5] as communication protocols or styles proved to be very effective tools for rapid prototyping. It should be noted that *only* REST was used for providing data *to* the application, as per Uckelmann, Harrison and Michahelles (2011). The prototype did not require two-way machine-to-machine communication, so COaP was not relevant to this system.

[2]https://storm.apache.org/
[3]https://store.spark.io/?product=spark-core
[4]http://ionicframework.com/
[5]https://www.meteor.com/ddp

An obvious downside of this approach is that a framework (like Meteor) may impose requirements to other dependencies in the application. In DAaaS, the main issue was that Meteor only supports the document database MongoDB[6] out of the box. There are several other stores better suited than MongoDB to store timeseries data, which was expected to be stored in DSaaS. Possibly better alternatives include TempoIQ[7] and InfluxDB[8]. Being required to use MongoDB for storage required a custom data structure to achieve acceptable performance, but that is outside the scope of this article.

Another important point to make about using established protocols, even if they (like Meteor's DDP) are not widely used outside of a small community, it may be easy to find third party libraries to help speed up development. For example, the real-time consumer graph used the library *asteroid*[9]. By defining a custom protocol with WebSockets, all communication must have been implemented by hand.

As long as there are not enough good all-purpose reference implementations with the proposed frameworks and protocol stacks, building something based on existing and well-defined protocols is easier. For rapid prototyping of a system, it seems best to prefer well-defined protocols and architectural styles like REST, and try to use existing frameworks for both client- and server-side applications. For commercial products, however, and especially if one aims to deliver several variations of the same product, service, or platform, exploring and using protocol stacks and frameworks developed specifically for the Internet of Things may be the best fit.

Several aspects of building a commercial application for actual use have been blatantly ignored in the development of DAaaS. Examples include security in both providing and consuming data; privacy, which has not been considered whatsoever (and rightfully so: the platform only stores and displays data in a custom fashion); and no error handling is implemented: if anything unexpected happens, the system will not do anything to restore state or shut down gracefully. These are all considerations to make which may differ from the regular Web application when introducing the aspect of Internet and Web of things.

No actual (big data) analysis of the data was performed by the prototype, leaving potential issues with this type of data unexplored by this article.

However, the possibly most important experience from developing the DSaaS application is that handling providers and consumers of the Internet and Web of Things just like any other type of client in the business logic of the application is tremendously helpful: if data from things needs to be transformed to fit a certain structure, then it should likely be transformed in the communication layer of the application before ever reaching the business logic.

As a final remark, HTTP/2[10] is on its way, and will certainly be an interesting player once released, allowing two-way communication and several asynchronous requests over the same connection. This may impact the need for COaP and WoT performance, create some disturbance in the effort to standardise WoT protocols, and certainly improve performance on the Web in general.

[6]https://www.mongodb.org/
[7]https://www.tempoiq.com/
[8]http://influxdb.com/
[9]https://github.com/mondora/asteroid
[10]https://tools.ietf.org/html/draft-ietf-httpbis-http2-17

5 Conclusion

We have seen that the current body of research on the Internet and Web of Things agrees that standardisation, full-stack research-oriented implementations, technology, and security are among the most important areas to look into in the future. Data Analysis as a Service attempts to address the first two of these issues, and is a small step on the way to bridging the gap. More focus must be directed at full-stack implementations of Internet and Web of Things-oriented applicatons, with special regard to separate use cases and domains. In particular, it should be interesting to see what matters in development of commercial products.

Utilising existing Web standards instead of developing the Internet and Web of Things as its own technology is going to be an important part of the process of simplifying the Internet of Things. We will probably require some new protocols as well – CoAP is a great example of this – but developing the WoT with the Web in mind will be crucial. At present, business needs and proposed technology, frameworks, and protocols are in conflict – but as more example implementations become available, this will hopefully change. Standardising protocols instead of having manufacturers implement custom means of communication is key to simplifying the Internet and Web of Things.

Security, privacy, cost, and maintenance of a distributed network such as the Internet of Things are still major considerations to make, and are certainly directions in which the academic community should go in the near future.

References

Paganelli, F., Turchi, S. & Giuli, D. (2014). A web of things framework for restful applications and its experimentation in a smart city. *IEEE Systems Journal*, 1–12.

Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G. & Dohler, M. (2013). Standardized protocol stack for the internet of (important) things. *IEEE Communications Surveys & Tutorials*, *15*, 1389–1406.

Shelby, Z., Hartke, K. & Bormann, C. (2014). The constrained application protocol (coap). Retrieved April 10, 2015, from https://tools.ietf.org/html/rfc7252

Uckelmann, D., Harrison, M. & Michahelles, F. (2011). *Architecting the internet of things.* New York: Springer.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 94–104.

Xu, L. D., He, W. & Li, S. (2014). Internet of things in industries: a survey. *IEEE Transactions on Industrial Informatics*, 2233–2243.