Data Analysis as a Service: an Infrastructure for Storing and Analyzing the Internet of Things

Martin Lehmann[2], Andreas Biørn-Hansen[2], Gheorghita Ghinea[1]
Tor-Morten Grønli[2] and Muhammad Younas[3]

[1]Brunel University, London, UK
[2]Westerdals Oslo ACT, Faculty of Technology, Oslo, Norway
[2] Oxford Brookes University, Oxford, UK

martin@westerdals.no, andreasb.nor@gmail.com,
tmg@westerdals.no, george.ghinea@brunel.ac.uk,
m.younas@brookes.ac.uk

Abstract. As the Internet of Things (IoT) is becoming an increasingly trendy topic both for individuals, businesses and governments, the need for academically reviewed and developed prototypes focusing on certain aspects of IoT are increasing as well. Throughout this paper we propose an architecture and a technology stack for creating real-time applications focusing on time-series data generated by IoT devices. The architecture and technology stack are then implemented through a proof-of-concept prototype named Office Analysis as a Service, DaaS, a data-centric web application developed using Meteor.js and MongoDB. We also propose a data structure for storing time-series data in a MongoDB document for optimal query performance of large datasets. One common research challenge in the IoT, *security*, is considered only briefly, and is of utmost importance in future research..

Keywords: Internet of Things, MongoDB, Data Analysis as a Protocol, Meteor, RESTful services, Reactive visualisation

1 Introduction

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 [6], and so the academic community must thoroughly address the area. Although 'IoT' was initially meant to describe a network of RadioFrequency ID-enabled devices, it has since been expanded to the following widely accepted definition [6]:

> *a dynamic global network infrastructure with self-configuring 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 [6]).

It becomes clear that the Internet of Things indeed encompasses all devices with a sensor, but there is also a second implication: the huge 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. [6] and Palattella et al. [4]. 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 paper.

Also important to mention is the Web of Things (WoT) [1]: the software layer on top of the Internet of Things. This paper 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. [4], which emphasises emerging industry alliances and IEEE/IETF working groups as the key to success. Finally, the pre-eminent concern of this paper is the gap of knowledge with regard to modelling and implementing complete IoT-oriented applications, as described by Paganelli, Turchi and Giuli [3].

This paper 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 Related Work

In this section we consider relevant literature and related work within the field. Xu et al. [6] contribute a major review of the current research on the Internet of Things (IoT). A very recent survey paper [6] 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 paper. Additionally, they propose a service-oriented architecture (SOA) style approach to the Web of Things. This approach is not considered by this paper. As mentioned in the introduction, Paganelli et al. [3] describes a lack of actually modelled and implemented applications as a major hole in the current research body. This paper also refers to a relatively large number of other papers proposing middleware and frameworks for designing applications in the Web of Things. However, Palattella et al. [4] claim 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 actually considered 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. [6] 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 paper, 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 paper.

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. [6] put it: the Internet of Things is still in its infancy.

3 Data Analysis as a Service (DSaaS)

A clear gap identified in the previous section is the lack 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, 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.

The current architecture and technology stack is the result of several iterations in which we experimented and prototyped in order to find the most well-fitting combination for our paper. We initially laid out a few requirements for the architecture and stack to support. Examples of such include the ability to rapidly prototype the artifact, support real-time data synchronization at some level, and it should fit into previously discovered challenges related to the IoT.

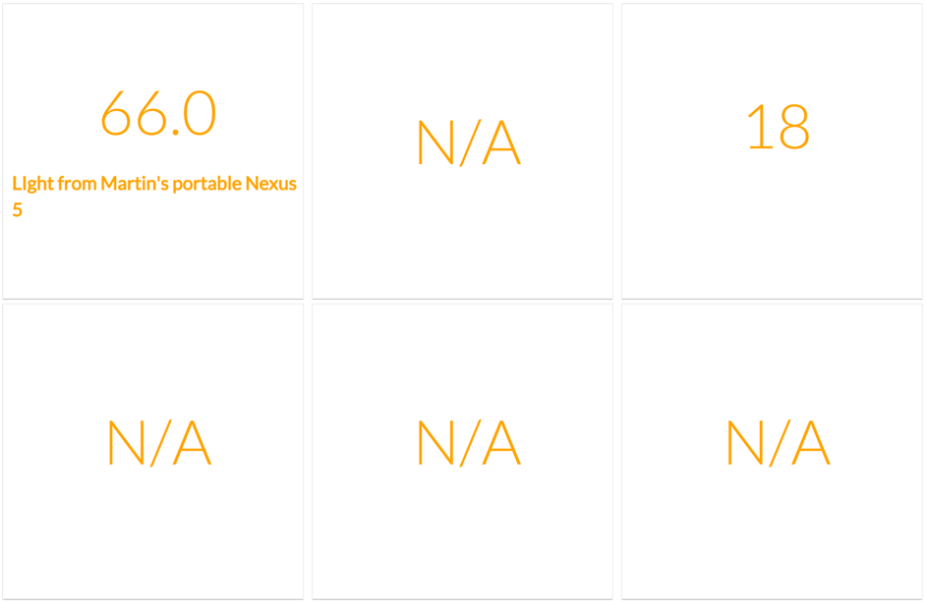

Fig. 1. The simple dashboard with two integrations

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 change feed for a specific resource to receive updates to the dataset in real-time. DSaaS also provides a very simple real-time dashboard (Fig. 1) for monitoring incoming data. Finally, it provides a management interface for customizing the dashboard and defining the *integrations* that can be displayed in the dashboard.

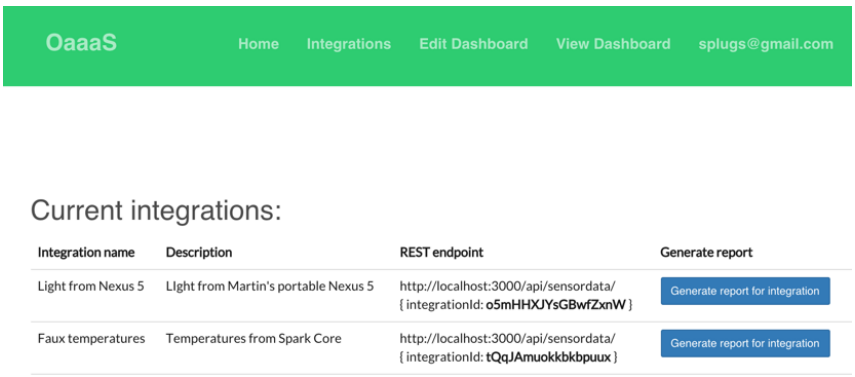

Fig. 2. Sample integrations

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

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 Fig. 2, creating an integration automatically generates a unique 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 change feed 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)[2], which is based on WebSockets. Because DDP's publish and subscribe-pattern (pub-sub) is agnostic [2] and not coupled with Meteor.js, DDP can be used to communicate between server-to-client, machine-to-machine, etc. This goes back to the interoperability aspect identified in several reviewed paper. It could naturally be possible to define a custom protocol with plain WebSockets, and that would enable building real-time graphs or custom dashboards for the data, or real-time analysis with for instance Apache Storm[3].

The prototype also includes three sample integrations/data providers (a Spark Core microcontroller[4], a native Android application listening for light values in the room using the light sensor on the mobile device, and a simple Ionic[5] 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 real-time graph for a single sensor.

4 Implemented Prototype Artefact

Our prototype, named Office Analysis as a Service (DaaS) consists of a web application where users can sign up and log in to the service, register new integrations (their own sensors), edit their dashboard, and view the dashboard to be displayed at a monitor or similar. The initial idea was to provide offices and workplaces with the ability to monitor their environments, and act on the resulting data. The end-product became rather general as it stores data from any source, being sensors or similar, as long as the data is in a given format, so the DaaS name is merely a thing of the past

There are various databases, like InfluxDB, KDB+ and KairosDB, exclusively developed to handle such data structures, but MongoDB comes bundled with Meteor.js, and is currently the only database fully supported by the Meteor Development Group.

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

Because of the tight coupling, it was decided to implement a time-series data structure into MongoDB instead of writing an adapter for Meteor.js to talk to InfluxDB or some other time-series-only database. Because MongoDB is a document database, we store data in *documents* and *collections* instead of rows and tables like in a traditional SQL database. For instance, we have implemented a collection named IntegrationData, where a document has the following properties:

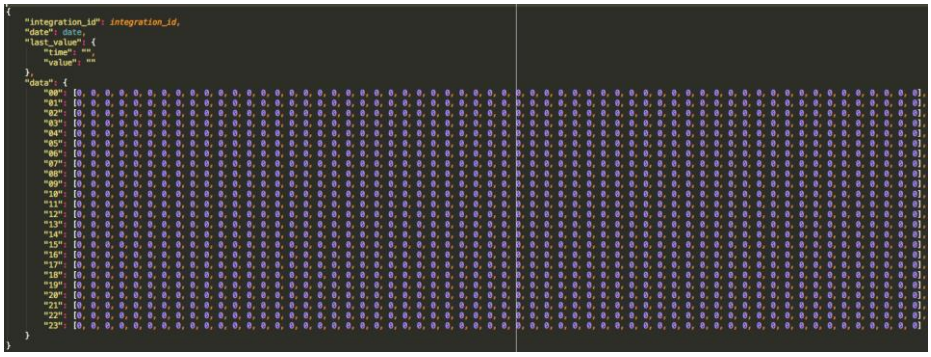

Fig. 3. A document data structure for time-series data in MongoDB.

`integration_ID`	The ID of an integration (physical sensor) to distinguish one sensor from another.
`date`	The date a given day in a given format.
`last_value`	A JavaScript object holding the latest value inserted into the document:
`-time`	the hour and minute (HH:MM) of an inserted value
`-value`	the value (numeric, string, bool)
`data`	A JavaScript object holding 24 arrays, one for each hour in a day. Each array has 60 indexes, one for each minute in that given hour.

Table 1. Properties of a time-series data document

This structure (Table 1) enables the client to quickly request the current data (`last_value` property) for real-time-display purposes, as well as for external services to integrate into DaaS to get time-series data (`data` property) for each day for each integration (`integration_id` property). Additionally, the size of the document is kept reasonable compared to the maximum size of 16 megabytes per document [2]. The main limitation with this approach is that one can only store one value per minute, else the previous value is overwritten, and if an integration's microcontroller halts and stops the data sending, the value for those minutes within the halted time frame will stay at 0. It is still the most optimal way we found to adapt parts of MongoDB's own advice on time-series storage into our service MongoDB based on read and write time, document size limitations and query optimization.

In order to prove parts of our current architecture, we developed an external (non-Meteor.js) example application connecting to our Meteor.js server instance through the DDP protocol. The example is a single HTML page graphing history data levering JavaScript, the D3 graph library, and the JavaScript library Asteroid for simple DDP connectivity. The DDP connection and MongoDB collection subscription (`getIntegrationDataForDDP`) with a parameter (`integration_id`) is configured like this

```
var asteroid = new Asteroid('oaaas.meteor.com');

asteroid.on('connected', function ()
{
    console.log('asteroid connected');
});

asteroid.subscribe('getIntegrationDataForDDP', 'HNcHW7wRKiw6MBEvE');

var lightCollection = asteroid.getCollection('IntegrationData');

var reactiveQuery = lightCollection.reactiveQuery({});
```

Fig. 4. Connecting to a Meteor.js instance with the Asteroid DDP connector

On change-events, an `asteroid.on('change', {})` event will fire, similar to the `asteroid.on('connected', {})` event, and serve the example application new, real-time data from the subscribed integration/sensor. This is in practice how the service, architecture and technology stack enables 3rd party developers, external services and more to integrate into our prototype and real-time aggregate on the time-series information we store. The prototype also provides RESTful non-real-time endpoints for externals to hook onto and use for more static purposes, like weekly Excel reports or similar. The implementation of this is based on Paganelli et al.'s [3] idea of treating each integration as a web resource and builds further on how to handle persisting time-series data in a document-database like MongoDB.

5 Discussion: experiences from developing DAaaS

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 customize what already exists. This experience differs from the main proposition in Palattella et al. [4], 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 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 [5]. The prototype did not require two-way machine-to-machine communication, so COaP was not relevant to this system.

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 (I.e. TempoIQ and InuxDB) better suited than MongoDB to store timeseries data, which was expected to be stored in DSaaS. Being required to use MongoDB for storage required a custom data structure to achieve acceptable performance.

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[7]. 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 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.

While no actual (big data) analysis of the data was performed by the prototype, leaving potential issues with this type of data unexplored by this paper, the proof-of-concept shows that, in its current state, it can connect Internet-enabled devices to our service via REST interfaces, persist the time-series data in a query-optimized fashion, and both real-time (DDP) and statically (REST) integrate into external services.

Because of DDP's agnostic communication-approach, it could be of interest to research on the protocol's ability to handle real-time machine-to-machine communication in constrained environments. Overall, the DDP protocol has proven itself as a potential standard for real-time data synchronization between client and

[6] https://www.mongodb.org/

[7] https://github.com/mondora/asteroid

server, and the REST paradigm for sending data between constrained environments (sensors via microcontrollers) to RESTful endpoints at a server.

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[8] 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 standardize WoT protocols, and certainly improve performance on the Web in general.

6 Conclusion

We have seen that the current body of research on the Internet and Web of Things agrees that standardization, 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 applications, 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 and upcoming technology advancements like HTTP/2 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. Standardizing 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.

7 References

1. Duquennoy, S., Grimaud, G. and Vandewalle, J.-J., 2009. The Web of Things: Interconnecting Devices with High Usability and Performance
2. Meteor (2015). DDP Meteor.com. [online] Available at:< https://www.meteor.com/ddp >.

[8] https://tools.ietf.org/html/draft-ietf-httpbis-http2-17

3. 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.
4. 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
5. Uckelmann, D., Harrison, M. & Michahelles, F. (2011). Architecting the internet of things. New York: Springer.
6. Xu, L. D., He, W. & Li, S. (2014). Internet of things in industries: a survey. IEEE Transactions on Industrial Informatics, 2233-2243.