Enabling Linked Data access to the Internet of Things

Timofey Ermilov
AKSW/BIS, Universität Leipzig
PO Box 100920, 04009 Leipzig, Germany
ermilov@informatik.uni-leipzig.de

Sören Auer
CS/EIS, Universität Bonn
Römerstraße 164, 53117 Bonn, Germany
auer@cs.uni-bonn.de

ABSTRACT

The term Internet of Things refers to the vision, that all kinds of physical objects are uniquely identifiable and have a virtual representation on the Internet. We present an approach for equipping embedded and smart devices on the Internet of Things with a Linked Data interface. The approach is based on mapping existing structured data on the device to vocabularies and ontologies and exposing this information as dereferencable RDF directly from within the device. As a result, all smart devices (e.g. tablets, smartphones, TVs) can easily provide standardized structured information and become first class citizens on the Data Web. A particular specific requirement when dealing with smart and embedded devices is resource constraints. Our evaluation shows, that the overhead introduced by equipping a device with a Linked Data interface is neglectable given modern software and hardware environments.

Categories and Subject Descriptors

H.3 [**Information Storage And Retrieval**]: General; H.3.4 [**Information Storage And Retrieval**]: Systems and Software—*Distributed systems*

General Terms

Measurement, Performance, Experimentation

Keywords

Data Web, Internet of things, Linked Data, Mobile

1. INTRODUCTION

The term Internet of Things [1] refers to the vision, that all kinds of physical objects are uniquely identifiable and have a virtual representation on the Internet. The unique identification can be realized using barcodes, RFIDs [18] or embedded systems and smart internet-enabled devices. In the former two cases the object itself can only identify itself and a virtual representation has to be hosted elsewhere. However, increasingly often some form of intelligence is embedded into the objects themselves (e.g. by integrating a system on a chip into a TV set or manufacturing equipment) or the object itself is a smart device (e.g. a smartphone or tablet PC). As a result, these devices can not only identify but also describe themselves by providing comprehensive information. There have been first attempts of integrating Web servers and hosting Web-accessible information within such devices (e.g. [5]). However, as we meanwhile complemented the Web of Documents with a Web of Semantic Data, information provided on the Internet of Things should be made available in standardized and semantically structured form as well.

In this paper, we present an approach for equipping embedded and smart devices with a Linked Data interface. The approach is based on mapping existing structured data on on the device to vocabularies and ontologies and exposing this information as dereferencable RDF directly from within the device. The technical architecture comprises a Web server running on the device, which serves content provided by a management service from a embedded triple store and device specific data stores. We implemented our approach for Android, which is an increasingly popular operating system not only for smartphones and tablet PCs, but also for smart TVs, navigation systems, cash registers and many other smart devices. Also, our implementation is easily adaptable for other Linux or Unix based embedded OS, such as *FritzOS*, *Firefox OS*[1] or *Sailfish OS*[2]. As a result, all smart devices can easily provide standardized structured information and become first class citizens on the Data Web.

Equipping smart devices with Linked Data interfaces has a number of advantages including:

- *Standardization.* Other data syndication and integration techniques are mostly proprietary and require integration at design time. With Linked Data interfaces, smart devices can expose, exchange and integrate data in ways unforeseen at design time.
- *Timeliness.* Since the data is directly exposed from the device where it originates, there are no delays related to data replication, caching etc. Persistence proxy services (similar to `purl.org`) can be used to maintain access to the data when devices are offline.
- *Privacy and data security.* Users' data is kept where

[1] `http://www.mozilla.org/en-US/firefox/partners/#os`
[2] `http://sailfishos.org/`

iiWAS2013, 2-4 December, 2013, Vienna, Austria.

it belongs (on their devices) and does not have to be centrally stored in order to be exchanged. Also, using FOAF+SSL and access control mechanisms data can be exposed in a fine-grained way.

A particular specific requirement when dealing with smart and embedded devices is resource constraints. Due to progress in miniaturization, memory and processing power is meanwhile not a constraining factor anymore for most applications. Power consumption on the other hand is a key aspect, when equipping devices with additional functionality. Hence, a particular focus of our implementation is limitation of the impact on power consumption and for this purpose and due to lack of existing standards in the area we develop a performance vs. power benchmarking methodology. The evaluation of our approach using this methodology shows, that the overhead introduced by equipping a device with a Linked Data interface is neglectable given modern software and hardware environments and moderate usage.

The article is structured as follows: We describe our approach for adding a linked data interface to smart devices in section 2. We discuss our implementation in section 3. The evaluation methodology is devised in section 4 and we also report corresponding results for our implementation in this section. We survey related work in section 5. We conclude in section 6 with an outlook on future work.

2. APPROACH: EMBEDDED LINKED DATA SERVER

The main goal of our approach is to enable any smart device (e.g. tablets, smart phones, TVs) to identify and describe itself by providing comprehensive information in accordance with the Linked Data principles. As a way to accomplish this goal, we propose the concept of an *Embedded Linked Data Server* (ELDS) that comprises a Web server, which hosts Web-accessible RDF information as Linked Data directly from within the smart device. The ELDS concept is based on an on-demand transformation of internal smart device data structures to RDF by using user-provided mappings.

2.1 Architecture

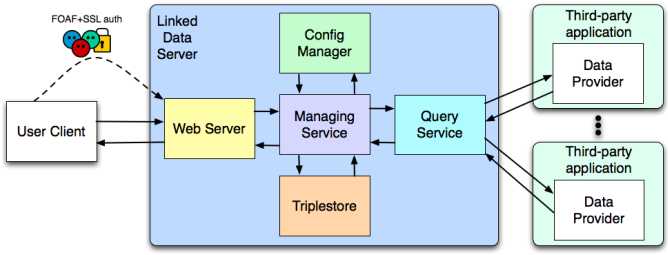

Figure 1: Overview of the Embedded Linked Data Server architecture.

The overall ELDS architecture is shown in Figure 1 and includes three layers:

1. *User layer* – comprises a user client,
2. *ELDS layer* – comprises ELDS internal components,
3. *Third-party application layer* – comprises third-party applications.

The separation in these three layers not only facilitates modularity and maintainability, but also aims at minimizing the impact on power consumption. Components of the user and third-party layer, for example, can easily be suspended thus minimizing the main memory and power consumption requirements.

The *User layer* includes user client application that can process RDF data and optionally a user WebID that is used for authentication and access control to the data provided by ELDS. *ELDS layer* includes all of the solution internal components. In particular: The *Web Server* component that receives requests from and sends replies to the user client. It also handles authentication and access control by utilizing FOAF+SSL protocol [11] if necessary. The *Managing Service* acts as main controller that directs all other internal components and manages most of the ELDS workflow. It receives request parameters from the Web Server, determines which data mapping configuration file should be applied, passes parameters to the Query Service and transforms structured data to RDF using the Triplestore component. The *Configuration Manager* component is responsible for loading and managing the data mapping configuration files provided by the user. The *Query Service* is responsible for fetching the data from third-party application data providers using a configuration object that is passed to it. The *Triplestore component* is used by the Managing Service to transform structured data that was received from the Query Service component into the requested RDF serialization. *Third-party application layer* includes external applications that can act as structured data providers inside of the smart device or object (e.g. Android contacts data provider).

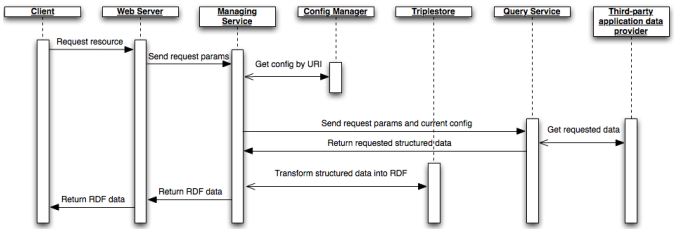

Figure 2: ELDS workflow during Linked Data access.

The basic ELDS workflow during Linked Data access is shown on Figure 2 and consists of following steps:

1. The client requests a specific resource description from ELDS on the smart device using an URI.
2. If ELDS requires authentication and enforces access control, the client must complete a FOAF+SSL authentication procedure (optional).
3. The Web Server component receives the request and passes its parameters to the Managing Component.
4. The Managing Component identifies configuration ID and fetches the required configuration object from Configuration Manager.

5. The configuration object together with other request parameters is sent to the Query Service.

6. The Query Service executes the query on third-party data providers applying the given parameters and returns results to the Managing Component.

7. The Managing Component transforms the results it has received from the Query Service into requested RDF serialization using the triple store component in accordance to the current configuration object.

8. The Managing Component passes resulting RDF serialization back to Web Server component.

9. The Web Server component returns resulting the RDF serialization back to client.

2.2 Use cases

There are a vast number of potential use cases for ELDS. Examples include (a) integrating data from an ELDS-based cash register into an ERP system, (b) autonomous weather stations directly publishing measurements, (c) exposing contact data in a distributed social network, (d) setting or publishing usage history and contexts from a smart TV or game console for content recommendation. We briefly outline the latter two use cases in the sequel.

The first use case where ELDS fits perfectly is using it as a part of Distributed Semantic Social Network (DSSN) [14]. Including ELDS in the DSSN architecture allows easy access to device internal data in a standardized non-proprietary way, e.g. location information or address book. Enforcing WebID authorization will help protect the data from unwanted access by third parties. This would enable usage scenarios, where a user would not need to contact his friend to ask for her location, but could just poll this information directly from her friend's smart device without the need for a central service. Similarly, when knowing that a user's friend has third-party contact data that the user is interested in, she could obtain the required contact data directly from smart device without bothering her friend (as long as she was granted access to this information).

A second use case is related to growing popularity of smart TVs and gaming consoles. For example, devices such as the Android-based *Google TV*[3] or the gaming console *Ouya*[4] can be easily turned into linked data providers. By exposing usage history from such devices (e.g. movies that user watched from Google TV, or games that user played from Ouya) in Linked Data format, it is possible to enable personalized recommendations in a non-intrusive way. After history information is accessible as Linked Data, history entries can be interlinked with existing databases like DBpedia or IMDB to enhance the performance of a recommendations system. It is also possible to display extracted history information to user using some form of user interface where she can specify whether she liked a specific entry or not thus enhancing recommendations.

3. IMPLEMENTATION: ANDROID LINKED DATA SERVER

We have implemented an Android specific version of ELDS – *Android Linked Data Server* (ALDS)[5]. We used existing Android implementations of two components as integral parts of ALDS. The Android version of *Jetty*[6] was used as Web Server and the *Androjena* library[7] as a triple store. We developed and integrated a *Configuration Manager* to dynamically load configuration files based upon JSON formatted files. A *Query Service* was developed to query third-party Android Content providers on demand.

Content providers manage access to structured (i.e. tabular) data on Android platform. Content providers are the standard interface that connects data from one process with code running in another process. Content providers can be addressed using a URI. Querying is performed by specifying the content provider URI, query conditions and columns that must be returned.

The best way to implement a long running service on the Android OS is to utilize the *Bound Service*. Running ADLS as a service allows to work in background without the need for any additional user interface or application running. This will also allow to throttle resource usage in favor of foreground tasks more easily. Another feature of the Android Bound Service is dynamic event handling. This can be used to dynamically update and reload configuration files by a user interface or third-party applications. The steps to add a new configuration file (or reload update file) are:

- Generate (or update) a JSON description file for a specific content provider,
- Place the JSON description file into the application configurations folder,
- Trigger the ALDS Configuration Manager to reload (either via the UI or Bound Service).

After receiving configuration object ALDS will expose a Content Provider that was described by given configuration using Linked Data. The internal Android structures are transformed into RDF using Androjena according to the description it has received from a configuration file.

Data Access and Mapping Configuration

To simplify creation and implementation of the ALDS configurations we decided to use simple JSON formatted text files. As JSON is a text-based open standard designed for human-readable data interchange such configuration files can be easily generated by both users and third-party applications. As an example, ALDS includes configuration file that transform basic contact information (i.e. name and phone number) using FOAF vocabulary to RDF. This example is shown on Listing 1.

The first step to follow when creating a configuration for ALDS is to define a URI of a content provider. This is done by specifying `provider_uri` string field. Optionally, custom prefixes can be defined to simplify latter creation of the bindings to the data. This is accomplished by specifying `rdf_prefixed` key-value array, where keys are prefixes and values are prefix URIs used during transformation. The next step is to define a set of columns that should be fetched from the Content provider as well as rules on how the column

[3] `http://www.google.com/tv/`
[4] `http://www.ouya.tv/`
[5] Available at: `https://github.com/AKSW/ALDS`
[6] `http://www.eclipse.org/jetty/`
[7] `https://code.google.com/p/androjena/`

data should be transformed to RDF representation. This is accomplished by specifying the `columns` array. It consists of objects that describe content provider columns and their data bindings to RDF. The objects that describe columns include three fields:

- *id* – column identifier in third-party Content provider,
- *name* – user-readable name (optional, can be blank),
- *predicate* – predicate used during data transformation to RDF, can use prefixes defined earlier, if `null` the column will be skipped.

The final step is to define rules which facilitate the generation of external URLs. This is done by specifying the `uri_generation` object. It includes two fields:

- *prefix* – defines a prefix that is used for current configuration (should be unique across all configurations),
- *values* – defines how the resource unique URI part is generated from specified column data values.

Basic workflow example..

Listing 1 is a configuration exposing phone contacts as Linked Data. Imagine a scenario where a user client requests an URL for Angela Merkel's contact information using:

```
http://device/phones/1912__Angela+Merkel
```

Following the workflow ALDS will determine the configuration ID from the given URL, which is `phones`. The configuration object for `phones` along with a request parameters string (i.e. "`1912__Angela+Merkel`") will then be passed to the Query Service. The Query Service splits the parameters string into separate values according to the configuration object. In this case the values `1912` and `Angela Merkel` are matched to columns `_id` and `display_name`. The Query Service then executes a query to the content provider that has the following URI:

```
content://com.android.contacts/data/phones
```

The query requests a specific contact with parameters that were extracted before and set of columns defined in configuration object, i.e.: `_id`, `display_name` and `data1`. After fetching the data the Query Service passes it to the Managing Component. The Managing Component then applies the transformation rules that are described in the configuration object. Specifically it binds data from all columns with defined predicated to the current resource URI. The resulting output can be seen in Listing 2.

```
1 @prefix foaf:<http://xmlns.com/foaf/0.1/>
      .
2
3 <http://device/phones/1912__Test+User>
4       foaf:name "Test User" ;
5       foaf:phone "+491761234567" .
```

Listing 2: Example output

4. EVALUATION

One of the most important things to consider regarding mobile and embedded device usage is power consumption. Thus the ALDS impact on power consumption of mobile devices is our main evaluation target. In addition, we measure the ALDS response time that influences the user experience during interaction with the ALDS service.

4.1 Evaluation methodology

Our evaluation methodology is inspired by existing power consumption research (e.g. [13] or [2]) and adopted to our specific needs (i.e. measuring impact on just one application instead of multiple and avoidance of low level voltage measurements). For a single evaluation run we picked a 2 hours timespan in order to truly observe the power consumption impact. During the evaluation period a 54 Mbit/s WLAN connection and a 3G GSM network connection were enabled and permanently maintained. Over the evaluation duration, the device was not touched, had its display turned off in order to increase the measurement precision of the ALDS impact on power consumption (except for the heavy front end load case). Measures of power consumption were taken using the *Battery Log* application[8]. A desktop computer running a simple evaluation script was acting as a client. The script executed requests with a set interval, verified response from ALDS and recorded the response time. In order to exclude a possible influence on WLAN capacity and performance by other devices, the only two devices connected to the WLAN were the mobile device under evaluation and the desktop computer running the evaluation script.

Before beginning the evaluation, a warm-up phase was performed for the ALDS and Android components by accessing RDF resources provided by ALDS three times without logging the results. That was required to prevent distortion of the evaluation results, since directly after the launch of the services it takes 3100 ms on average to get response from ALDS.

4.2 Evaluation testbed

Two different Android devices were used for evaluation. The first device is the mobile phone Samsung Galaxy i9003 SL running stock Android v2.3.6 (Gingerbread). It features the TI OMAP 3630 chipset, comprising a 1 GHz single core Cortex-A8 CPU, 2 GB of internal storage complemented by 16 GB SD card, 478 MB RAM and a 1650 mAh Li-Ion battery. The device was in constant use since June 2011, which means that battery was 23 month old at the time of evaluation. The second device is the tablet Smartbook Surfer 360 MN10U running custom Android 4.1.2 (JellyBean). It features the Nvidia Tegra 250 chipset, comprising a 1 GHz dual core Cortex-A9 CPU, 512 MB of internal storage complemented by 16 GB SD card, 512 MB RAM and 3300 mAh Li-Ion battery. The device was not used before performing evaluation, which means that battery was completely new at the time of evaluation. As the test data set we had used contact book from one of author's personal devices. The size of the contacts data set is 7.43 mb. The memory requirements are below 32 mb which should work for both test devices.

Power consumption and average response time were measured in the following six scenarios:

[8] `https://play.google.com/store/apps/details?id=kr.hwangti.batterylog`

```
provider_uri: "content://com.android.contacts/data/phones",
rdf_perifxes: {
  "foaf": "http://xmlns.com/foaf/0.1/"
},
columns: [
  {
    id: "_id",
    name: "id",
    predicate: null
  },
  {
    id: "display_name",
    name: "username",
    predicate: "foaf:name"
  },
  {
    id: "data1",
    name: "number",
    predicate: "foaf:phone"
  }
],
uri_generation: {
  prefix: "phones",
  values: ["_id", "display_name"]
}
```

Listing 1: Example configuration which maps the contacts provider to the FOAF vocabulary.

- **S1:** Device without ALDS
- **S2:** Device with ALDS in stand-by mode
- **S3:** Device with ALDS with 1 request every 30 minutes
- **S4:** Device with ALDS with 1 request every 5 minutes
- **S5:** Device with ALDS with 1 request every 1 second
- **S6:** Device with ALDS with 5 concurrent requests per 1 second

For each scenario we performed all measurements 5 times in order to average out power consumption variation (e.g. due to background OS processes or network overhead activity).

4.3 Benchmarking results

The results of the power consumption evaluation are show in Figure 3. As it can be seen, ALDS in idle state (S2) as well as periodic requests with 30 minute (S3) and 5 minute (S4) frequency have no observable impact on battery consumption. Periodic requests with 1 second frequency (S5) have a small impact. The difference to S1 (no ALDS installed) is in range of 3-7% points of the charge. Five concurrent requests per 1 second (S6) have the worst impact on power consumption. The difference to S1 (no ADLS installed) is in range of 10-14% point of the charge.

Due to bugs encountered in the Androjena library, in the scenario with 5 concurrent requests per second (S6), ALDS has stopped sending proper responses on average 2.3 times per evaluation duration. At that point, the service had to be restarted and the evaluation started from the beginning. Because it was impossible to go through the whole 2 hour evaluation cycle, we used two or more iterations to go through the complete 2 hours timespan. Our observation shows that a possible cause of such behavior could be due to Androjena not freeing resources properly (or enough) after usage thus exceeding the limited Android VM heap size (32 MB for devices used in evaluation) and causing the library to stop functioning.

Because measuring only power consumption by ALDS was not possible or the difference with already existing evaluation scenarios was insignificant, we measured only the average response times in the following additional scenarios:

- **S7:** Device with ALDS with 1 request every 30 seconds
- **S8:** Device with ALDS with 1 request every 10 seconds
- **S9:** Device with heavy front end load and ALDS with 1 request every 1 second

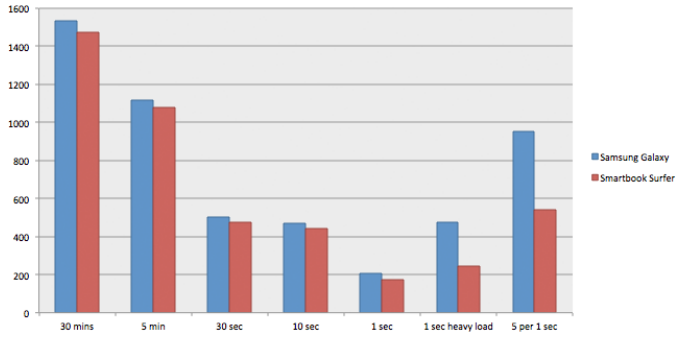

Figure 4: Average ALDS response time per linked data access (in ms).

The results of response time evaluation are shown in Figure 4. As can be seen, ALDS response time differs depending on request frequency. In the scenario with one request per 30 minutes (S3) the average response time for the Samsung

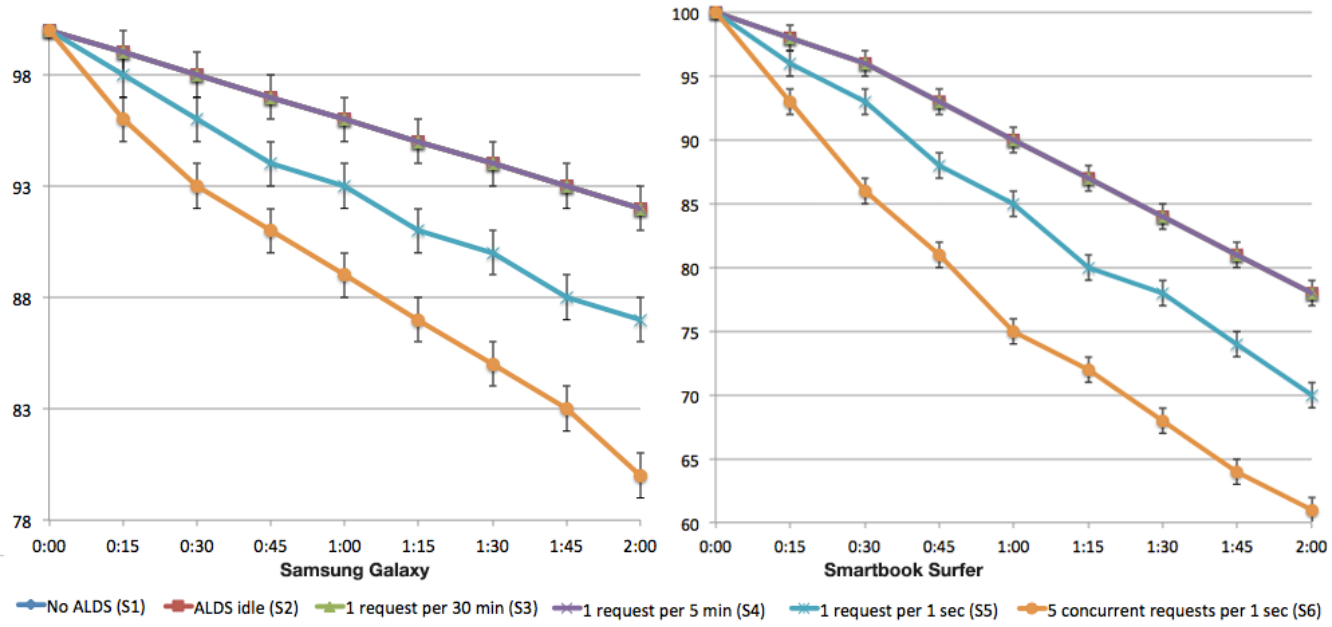

Figure 3: ALDS power consumption in terms of battery charge percentage (y-axis) over time (x-axis).

Galaxy is 1,535 ms and for the Smartbook Surfer 1,474 ms. This long response time is caused by Android suspending ALDS and the third-party content provider to save power. Most of the response time can be attributed to waking up those components from suspension mode.

In the scenario with one request per 5 minutes (S4) the average response time for the Samsung Galaxy is 1,116 ms and for the Smartbook Surfer 1,078 ms. Also here, the response time is high due to suspending ALDS and the third-party content provider. However, since requests are performed more frequently, ALDS is not suspended completely, but only partially (i.e. web server is still running, but the query component is woken up upon requests).

In the scenarios with one request per 30 seconds (S7) and 10 seconds (S8) the average response times for the Samsung Galaxy are 502 ms and 472 ms respectively, for the Smartbook Surfer 473 ms and 442 ms respectively. In these cases, ADLS is not suspended at all. While content providers can be suspended completely or partially depending on their architecture. For example, if a content provider stores parts of the data using different storage containers, one of the routes can be suspended.

In the scenario with one request per 1 second (S5) the average response time for the Samsung Galaxy is 205 ms and for the Smartbook Surfer 176 ms. This case is optimal in terms of response time, since ALDS and third-party content providers are not being suspended at all due to frequent requests.

In the scenario with one request per 1 second with heavy front end load (S9) the average response time for the Samsung Galaxy is 478 ms and for the Smartbook Surfer 245 ms. During the heavy front end load scenario, ALDS was running in background, while the device itself was used to navigate through a series of memory, CPU and bandwidth intensive applications. Applications include: Instagram (browsing images), Pinterest (browsing images), Facebook (browsing feeds, photos, using chat), Play Store (browsing and applying updates to installed apps), YouTube (watching videos), Cut The Rope: Time Travel (playing game). During the benchmark there was from a user perspective no noticeable impact observable on active application performance, application switching, network speed or other features. The response time increased in comparison to one request per 1 second case by approximately a factor 2.5 for the Samsung Galaxy and by approximately a factor 1.4 for the Smartbook Surfer. This is due to Android prioritizing applications running in foreground compared to background services. The difference between response times between the Samsung Galaxy and the Smartbook Surfer can be explained by the dualcore processor of the Smartbook Surfer that allows better handling of multitasking.

In the scenario with five concurrent requests per 1 second (S6) the average response time for the Samsung Galaxy is 952 ms and for the Smartbook Surfer 542 ms. The increase in response time in this case is caused by Androjena conversion delay, meaning that Androjena cannot process transformation of structured data into RDF fast enough. This might as well be related to the heap size issue mentioned before. The difference between response times of Samsung Galaxy and Smartbook Surfer can again be explained by the dualcore processor of the Smartbook Surfer that allows Androjena to perform the conversion slightly faster.

As our evaluation shows, the ALDS impact on device power consumption while answering infrequent requests (requests every 30 and 5 min, DSSN use case) can be considered insignificant. On other hand, if ALDS is going to be used intensively, an external power source would be required after several hours of work.

5. RELATED WORK

	Raw data	*Structured data*	*Semantic data*
RFID	Bridging Physical and Virtual Worlds: Complex Event Processing for RFID Data Streams [18]	-	-
Sensors	Fjording the stream: An architecture for queries over streaming sensor data [7]	A Pipelined Framework for Online Cleaning of Sensor Data Streams [6]	-
Embedded	MEDIC: Medical embedded device for individualized care [20]	Code generation techniques for developing light-weight XML Web services for embedded devices [17]	Semantic Web services for smart devices in a "global understanding environment" [12]
Mobile / Smart	Mobile landscapes: using location data from cell phones for urban analysis [10]	XMIDDLE: A Data-Sharing Middleware for Mobile Computing [8]	*Enabling Linked Data access to the Internet of Things*
Desktop / Server	-	Bigtable: A Distributed Storage System for Structured Data [3]	Jena: a semantic Web toolkit [9]

Figure 5: Comparison of selected existing devices and data provider approaches.

To the best of our knowledge the work presented in this article is the first tackling Linked Data exposure and access from smart devices and its impact on power consumption. On a more general level, related work can be divided into several separate categories by types of devices and data they expose. An overview of related work is shown in Figure 5.

Types of data comprise:

- *Raw data* – data that does not have a pre-defined data model,
- *Structured data* – data that have a pre-defined data model,
- *Semantic data* – data that have a pre-defined data model and includes additional semantic information.

With regard to device types we can distinguish in order of increasing processing power RFID chips, sensors, embedded systems, smart/mobile devices and traditional desktop/server computing.

The first device type is simple RFID tagging systems. Research in this category is focused on RFID raw data access. For example, [18] takes an event-oriented approach to process RFID data, by devising RFID application logic into complex events. Since RFID cannot really provide anything aside from raw data (tag information), there is no research on structured or semantic data. However, RFIDs can be used for identifying objects and (semantic) object descriptions and related information can be stored elsewhere. An example of such an approach is *Semantic Space* [19] that incorporates Semantic Web technologies into pervasive computing environments. Or the museum tour guide [4] that features a semantic web rule reasoning engine that enables visitor-oriented services to identify relevant sources of contextual information.

The second device type is sensors. Since sensors are usually not used one at a time, raw data access researches focus mainly on data aggregation from networks of sensors. For example [7] presents the Fjords architecture for managing multiple queries over many sensors. Since raw data access is not the best option when dealing with large amount of data fetched from networks of sensors, there is research focusing on exposing data fetched from sensors as structured data. For example, *Extensible receptor Stream Processing* (ESP) [6], an extensible framework for cleaning the data streams produced by physical receptor devices.

The third device type are embedded devices and systems. Raw data research in this area is mainly focused on receiving and processing raw data from external sensors for later exposure to third-parties. For example, the *medical embedded device for individualized care* (MEDIC) [20], enables sensor management and disease prediction capabilities using commercially available microelectronic components, sensors and conventional personal digital assistant. Structured data research in this area is mainly focused on exposing data using a web service approach with typically XML data formatting. For example, [17] presents specialized code generation techniques and runtime optimizations for developing light-weight XML Web services for embedded devices. Semantic data research in this area is mainly focused on smart industrial devices, robots or any other embedded devices or systems that might need special services for condition monitoring, information provisioning, remote diagnostics, maintenance support, etc. An example is the *Global Understanding Environment* [12] that specifies main requirements to Web services that automatically follow up and predict the performance and maintenance needs of field devices. There is as well research like [5] that demonstrates an approach similar to Web 2.0 mashup applications for integrating real-world devices to the Web, allowing them to be easily combined with other virtual and physical resources.

The fourth device type is mobile and smart devices. Raw data research in this area is mainly focused on exposure of raw data from location based services for latter analysis. For example, [10] aims to review and introduce usage of Location Based Services to the urban planning community as powerful tool for urban analysis. Structured data research in this area is most common topic with regards to data access and has a lot of publications available. For example, *XMIDDLE* [8] that enables the transparent sharing of XML documents across heterogeneous mobile hosts, allowing on-

line and off-line access to data. Semantic data access for this type of devices, which is at the core of the work presented in this article, did not seem to be tackled by the research community yet.

The last device type is traditional desktop and server computing. No raw data research is available under this category as there is no need to work on such low level. Structured data under this category has been main research field for last years and presented with set of big projects and publication. For example, *Bigtable* [3] that is a distributed storage system for managing structured data that is designed to scale to a very large size. Semantic data under this category is developing rapidly and presented with a set of big projects and publication. For example, *Jena* [9] that is a Java application programming interface aimed to make it easier to develop applications that use the semantic Web information model and languages.

Additionally, RDB2RDF transformation research can be considered related to the presented approach. However, since Android content providers are very simplistic, we decided to follow our own mapping strategy. The RDB2RDF field is quite dynamic since the prevalent way to store data at the moment is in relational databases. Research in this field includes approaches like *SparqlMap* [16], a SPARQL-to-SQL rewriter based on the specifications of the W3C R2RML working group. A more in-depth overview can be found in a survey on knowledge extraction approaches from structured sources such as relational databases, XML and CSV [15].

6. CONCLUSION AND FUTURE WORK

In this paper, we presented an approach for equipping embedded and smart devices with Linked Data interfaces. Our approach is based on mapping structured data hosted on the device to RDF and exposing this data as Linked Data using an embedded webserver. Our implementation is currently implemented for Linux and contains some Android-specifics (i.e. using content providers). However, it is easily transferable to other embedded platforms. Android's content providers, for example, are simple tabular SQLite tables, which can be used in a similar fashion on other systems. We also showed with a newly developed benchmark methodology, that device power consumption does not increase significantly until Linked Data is retrieved frequently (>1 request per second). We argue, that this resource demands can be accommodated by most mobile use cases and power consumption is not an issue for stationary smart devices (e.g. TVs). We expect Android and similar OS to be deployed on more and more smart devices ranging from watches, smartphones, routers, devices with displays (e.g. refrigerators), cash registers and many other device categories currently not even yet available. As a result, the Web and Internet being accessed from desktop computers will loose importance compared to these novel usage scenarios. With ALDS we made a first step towards extending the Web of Data towards a Data Internet of Things, which comprises these scenarios.

In future work, we plan to extend our approach along several dimensions: We plan to implement smart caching for data fetched from a content provider, so that subsequent request do not require re-retrieving and processing data from a content provider. We aim to support complex configurations for queries across several content providers and to allow the execution of SPARQL queries. In order to improve availability we plan to realize a hybrid provider approach, where the server has access to client data and responds if the client is not available.

Acknowledgment

This work was supported by a grant from the European Union's 7th Framework Programme provided for the project LOD2 (GA no. 257943).

7. REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[2] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293. ACM, 2009.

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[4] S.-C. Chou, W.-T. Hsieh, F. L. Gandon, and N. M. Sadeh. Semantic web technologies for context-aware museum tour guide applications. In *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, volume 2, pages 709–714. IEEE, 2005.

[5] D. Guinard and V. Trifa. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, 2009.

[6] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. A pipelined framework for online cleaning of sensor data streams. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 140–140. IEEE, 2006.

[7] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 555–566. IEEE, 2002.

[8] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. Xmiddle: a data-sharing middleware for mobile computing. *Wireless Personal Communications*, 21(1):77–103, 2002.

[9] B. McBride. Jena: A semantic web toolkit. *Internet Computing, IEEE*, 6(6):55–59, 2002.

[10] C. Ratti, S. Williams, D. Frenchman, and R. Pulselli. Mobile landscapes: using location data from cell phones for urban analysis. *ENVIRONMENT AND PLANNING B PLANNING AND DESIGN*, 33(5):727, 2006.

[11] H. Story, B. Harbulot, I. Jacobi, and M. Jones. Foaf+ ssl: Restful authentication for the social web. In *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*, 2009.

[12] V. Terziyan. Semantic web services for smart devices in a “global understanding environment”. In *On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops*, pages 279–291. Springer, 2003.

[13] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50. ACM, 2012.

[14] S. Tramp, P. Frischmuth, T. Ermilov, S. Shekarpour, and S. Auer. An architecture of a distributed semantic social network. *Semantic Web*, 2011.

[15] J. Unbehauen, S. Hellmann, S. Auer, and C. Stadler. Knowledge extraction from structured sources. In S. Ceri and M. Brambilla, editors, *Search Computing - Broadening Web Search*, volume 7538 of *Lecture Notes in Computer Science*, pages 34–52. Springer, 2012.

[16] J. Unbehauen, C. Stadler, and S. Auer. Accessing relational data on the web with sparqlmap. In *JIST*, 2012.

[17] R. van Engelen. Code generation techniques for developing light-weight xml web services for embedded devices. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 854–861. ACM, 2004.

[18] F. Wang, S. Liu, P. Liu, and Y. Bai. Bridging physical and virtual worlds: complex event processing for rfid data streams. In *Advances in Database Technology-EDBT 2006*, pages 588–607. Springer, 2006.

[19] X. Wang, J. S. Dong, C. Chin, S. R. Hettiarachchi, and D. Zhang. Semantic space: An infrastructure for smart spaces. *Computing*, 1(2):67–74, 2002.

[20] W. H. Wu, A. A. Bui, M. A. Batalin, L. K. Au, J. D. Binney, and W. J. Kaiser. Medic: Medical embedded device for individualized care. *Artificial Intelligence in Medicine*, 42(2):137–152, 2008.