Querying Dynamic Datasources with Continuously Mapped Sensor Data[*]

Ruben Taelman, Pieter Heyvaert,
Ruben Verborgh, and Erik Mannens

Data Science Lab (Ghent University - iMinds)
firstname.lastname@ugent.be

Abstract. The world contains a large amount of sensors that produce new data at a high frequency. It is currently very hard to find public services that expose these measurements as dynamic Linked Data. We investigate how sensor data can be published continuously on the Web at a low cost. This paper describes how the publication of various sensor data sources can be done by continuously mapping raw sensor data to RDF and inserting it into a live, low-cost server. This makes it possible for clients to continuously evaluate dynamic queries using public sensor data. For our demonstration, we will illustrate how this pipeline works for the publication of temperature and humidity data originating from a microcontroller, and how it can be queried.

Keywords: Linked Data, Linked Data Fragments, RML, SPARQL, dynamic data

1 Introduction

Countless sensors that are connected to the Internet produce various streams of continuous data. Even though there are many sensors, no public interfaces are available which these measurements can be consumed as dynamic Linked Data [2], which is essential when these measurements need to be linked to related information.

Dynamic Linked Data can be published through RDF Stream Processing (RSP) query engines, like C-SPARQL [1] and CQELS [9]. These RSP query engines are used to publish this sensor data using an extended SPARQL [7] endpoint that enables continuous querying over this data. These engines require expensive machines to host such data on the Web, because queries can be complex, are continuous and originate from an unlimited number of clients. These expensive solutions limit the development of sensor-based applications [4].

In this work, we present a continuous pipeline for publishing dynamic sensor data using a low-cost Triple Pattern Fragments (TPF) [11] server. This publication makes it possible for evaluating queries continuously at the client using a TPF Query Streamer [10]. In order to demonstrate this pipeline, we publish and query the continuous data from a temperature and humidity sensor. In the demonstration, we will show the different steps conducted on the data, starting from the raw measurements until the final query results.

[*] The described research activities were funded by Ghent University, iMinds, the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research Flanders (FWO Flanders), and the European Union.

2 Architecture

Our continuous pipeline for publishing sensor data consists of a data reader, mapper and publisher. These components connect a sensor with a dynamic data source, as can be seen in Fig. 1. The reader receives data from a sensor. The mapper converts this data to RDF using the RDF Mapping Language (RML) [6]. Finally, the publisher inserts the dynamic RDF triples into a TPF server. These three components will be explained hereafter.

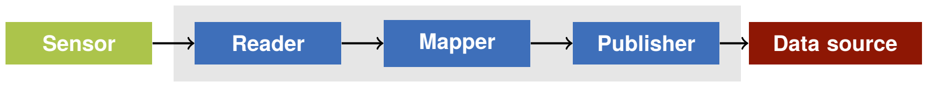

Fig. 1: Overview of the continuous pipeline for publishing sensor data.

2.1 Reader

For this demonstration, we use a Tessel[1] microcontroller to capture sensor data from the environment. Tessel provides a Node.js[2] module that can be used to listen to sensor measurements from the device using JavaScript events. JSON objects are emitted for each measurement at a configurable rate. These objects contain all required data to describe each measurement.

2.2 Mapper

The mapper component [8] is responsible for converting the JSON output that it receives from the reader component to RDF. This conversion is done using the RMLMapper[3]. For each measurement, the RMLMapper reads the RML mapping file, maps the JSON object to RDF and forwards the result to the publisher component. The mapping file depends on the type of sensor that is being used, because not all sensors output the same data in the same structure and format and the data is not necessarily annotated using the same ontologies.

2.3 Publisher

For the final step in the continuous pipeline, the publisher component takes the measurement data in RDF and converts it to *dynamic data*. The publisher does this by adding a time annotation to it, as we described in previous work [10]. We chose to represent our time annotations as expiration times and we only keep the latest version of each measurement, because for this demonstration only the latest sensor measurements are needed. Our system alternatively also supports storing all measurement versions, which enables historical analysis. We serialized our annotations as RDF graphs [5], because we showed that this is the most efficient method for time annotation [10]. Finally, each time-annotated measurement is inserted into the TPF server and the previous measurement for that sensor is removed.

[1] `https://tessel.io/`
[2] `https://nodejs.org/`
[3] `https://github.com/RMLio/RML-Mapper`

3 Demonstration Overview

We used our pipeline to publish data that is being created by a Tessel sensor that reads temperature and humidity data. The reader component starts by emitting measurements in JSON, like the following.

```
{ "humidity"   : { "value":47.0815124512 },
  "device"     : "TM-00-04-f000da30-0062473d-20a82586",
  "module"     : "climate",
  "temperature": { "value":30.0167749023 } }
```

In order to add semantical value to the measurements, the mapper component maps each JSON object to RDF.

```
<http://www.example.com/TM-00-04-f000da30-0062473d-20a82586>
  a <https://w3id.org/saref#Sensor> ;
  <http://vocab.datex.org/terms#humidity> "47.0815124512"^^xsd:double ;
  <http://dbpedia.org/ontology/temperature> "30.0167749023"^^xsd:double .
```

Next, the triples are time-annotated by the publisher component to make the triples valid up until a certain time. This is represented with the TriG [3] serialization.

```
_:g12 {
  <http://www.example.com/TM-00-04-f000da30-0062473d-20a82586>
    a <https://w3id.org/saref#Sensor> ;
    <http://vocab.datex.org/terms#humidity> "47.0815124512"^^xsd:double ;
    <http://dbpedia.org/ontology/temperature> "30.0167749023"^^xsd:double .
}
_:g12 <http://example.org/temporal#expiration> "2016-06-27T09:48:47.808Z"^^<
     xsd:dateTimeStamp> .
```

After the data is published, it can be consumed by clients. We use the client-side TPF Query Streamer to continuously evaluate the following SPARQL query in order to fetch the published measurements from our pipeline.

```
PREFIX dat: <http://vocab.datex.org/terms#>
PREFIX saref: <https://w3id.org/saref#>
PREFIX dbp: <http://dbpedia.org/ontology/>

SELECT ?temperature ?humidity WHERE {
    ?sensor a saref:Sensor;
            dat:humidity ?humidity;
            dbp:temperature ?temperature.
}
```

This query selects the temperature and humidity data from all sensors. The TPF Query Streamer is able to interpret this query as being dynamic, so it will automatically re-evaluate the query from the moment new measurements become available. The query can for example show the following intermediary result.

```
Temperature: "30.0167749023"     Humidity: "47.0815124512"
```

These continuously updating query results could, instead of printing it as text, be used as input to various types of applications.

During the demo, we will show the different steps conducted on temperature and humidity data measurements by the different components of the pipeline and the query results. These steps will be shown as continuously updating log outputs for each component. The Tessel microcontroller will be part of the demonstration, this will make it possible for users to modify the temperature or humidity and immediately see the changes in the query output. We recorded a screencast[4] containing these live logs for temperature and humidity data. The source code of our pipeline prototype can be found at `https://github.com/mmlab/demo-tessel-continuous-datasource`.

The demonstration illustrates the simplicity of hosting any kind of sensor data using this technique. For future work, this solution can be improved and extended to support all kinds of sensors and sensor streams. The pipeline could be extended to support composing different pipelines and streams in order to create more complex dynamic data.

With this demonstration we aim to show that publishing dynamic Linked Data does not have to be complex or costly. We hope to trigger more efforts for making dynamic data publicly available, in order to promote the development of more dynamic applications.

References

1. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying RDF streams with C-SPARQL. SIGMOD Rec. 39(1), 20–26 (Sep 2010)
2. Berners-Lee, T.: Linked Data (July 2006), `http://www.w3.org/DesignIssues/LinkedData.html`
3. Bizer, C., Cyganiak, R.: RDF 1.1 TriG. Rec., W3C (Feb 2014), `https://www.w3.org/TR/trig/`
4. Corcho, O., García-Castro, R.: Five challenges for the Semantic Sensor Web. Semantic Web 1(1, 2), 121–125 (2010)
5. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: Concepts and abstract syntax. Recommendation, W3C (Feb 2014), `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`
6. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A generic language for integrated rdf mappings of heterogeneous data. In: LDOW (2014)
7. Feigenbaum, L., Todd Williams, G., Grant Clark, K., Torres, E.: SPARQL 1.1 protocol. Rec., W3C (Mar 2013), `http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/`
8. Heyvaert, P., Taelman, R., Verborgh, R., Mannens, E.: Linked Sensor Data generation using queryable RML mappings. In: Proceedings of the 2016 International Semantic Web Conference Posters & Demonstrations Track (2016), accepted for publication
9. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and Linked Data. In: The Semantic Web–ISWC 2011, pp. 370–388 (2011)
10. Taelman, R., Verborgh, R., Colpaert, P., Mannens, E., Van de Walle, R.: Continuously updating query results over real-time Linked Data. In: Proceedings of the 2nd Workshop on Managing the Evolution and Preservation of the Data Web (May 2016)
11. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. Journal of Web Semantics 37–38, 184–206 (2016)

[4] `https://vimeo.com/172409187`