Converting and transfering data from another IoT platform to FIWARE: case Electric bus

Otto Hylli

May 26, 2020

Contents

1 Introduction

This report describes how measurements collected from a couple of City of Tampere electric buses to the commercial IoT-Ticket platform, developed by Wapice, were transfered to a FIWARE platform. The tool developed for the data conversion and transferance is described and lessons and experiences about sending data to FIWARE are discussed. The report assumes that the reader is familiar with FIWARE. In particular its Orion and Quantumleap components: what they are and how they are used. The reader should also be familiar with FIWARE's data modeling principles: the entity based data model and how, for example, timestamps are handled.

2 Background

This section explains the bus data, how it is stored to IoT-Ticket and what kind of FIWARE platform was used.

2.1 Data

Data is collected from 4 electric buses and one hybrid bus operating on the same bus line. Overall 18 different measurements are available. Measurements collected include speed, location as latitude and longitude, battery charge state percent, power, energy consumed, brace air pressure and door status. However not all measurements are available for all buses. Most notably only speed and location are available from the hybrid bus due to issues with configuring the data collection hardware. The measurements are sent to the IoT-Ticket in real time through Wapice's proprietary hardware which is connected to the bus systems. The update interval for the measurements varies. Some measurements such as charge state and door status are updated about every few seconds while some others such as location and speed are updated about every second. Measurement timestamps are recorded in microsecond precision. Measurements apart from latitude and longitude are updated independently from each other so they are unlikely to share exactly the same timestamp.

2.2 Data source

The buses are represented as sites in IoT-Ticket. Each measurement is saved to a datanode under the site. A datanode has a name, optional unit, and measurements saved as value timestamp pairs. Timestamps use the Unix time in microsecond precision i.e. a timestamp is an integer counting time from the Unix epoch 1970-01-01 00:00 UTC in microseconds.

The IoT-Ticket platform offers a REST API for getting the measurements. Relevant API endpoints for transfering the data are the endpoint, which lists all datanodes for a site, and the endpoint which gives measurements for a datanode from the given time period. The API uses the JSON format. Below is an example of the format used to give measurement values for a datanode. In each item v gives the measurement value and ts is the timestamp.


```
{
   "type": "processData",
   "limit": 10000,
   "order": "ascending",
   "begin": 1571742019797515,
   "end": 1571742029797515,
   "items": [
      {
         "type": "num",
         "ts": 1571742021712393,
```

```
            "v": 848
        },
        {
            "type": "num",
            "ts": 1571742022713517,
            "v": 848
        },
        {
            "type": "num",
            "ts": 1571742026716804,
            "v": 848
        },
        {
            "type": "num",
            "ts": 1571742027716926,
            "v": 848
        }
    ]
}
```


2.3 The FIWARE platform

The FIWARE platform, the data was collected to, consists of the Orion context broker, Quantumleap measurement history component and Grafana data visualization component. All of them are behind a NGINX web server which works as a reverse proxy and manages authentication with HTTP header based API keys.

3 Converting and transfering to FIWARE

This section explains what kind of FIWARE data model was created for the bus measurements and what kind of tool was developed for converting the data according to the model and sending it to FIWARE.

3.1 Data model

The existing FIWARE data models were not suitable for the electric bus data since they did not have attributes for all measurements. Thus a custom data model had to be created. However the existing Vehicle data model was used as a basis for this data model. It has attributes for some of the bus measurements such as speed, location and distance travelled. Attributes for the missing measurements such as power, battery charge state and door status were added to the Vehicle entity. The data model specification is available from the conversion tool's GitHub repository.

3.2 Electric bus collector tool

The requirements for the data collector tool was that it should be able to both collect historical data from IoT-Ticket, e.g. measurements from the previous month, and continuously collect new measurements as they arrive to IoT-Ticket. As is typical for a FIWARE solution the newest measurements should be available from Orion and historical measurements from the history component which, in this case, is Quantumleap. The tool was developed with Python. It is available from its GitHub repository.

Information about how to convert the IoT-Ticket measurements to FIWARE entities was defined in a configuration file. It lists the ids of IoT-Ticket sites containing the bus measurements and information about how to map datanodes to entity attributes. Another configuration file defines what measurements are collected. This is defined with a start and end date as follows:

- If start and end dates both are provided measurements between them are collected and the tool exits when done. Both dates should be in the past.
- If only start date is given the tool starts collecting historical measurements from that date until it reaches the current moment and starts collecting measurements continuously in near real time as they arrive.
- If no dates are provided continuous real time collection is started.

When started the tool first gets the list of datanodes for each bus site. Then it creates entities to Orion for each bus with static attributes if the entity does not already exists. Static attributes includes those dentity attributes whose value does not change and is not fetched from IoT-Ticket. This includes for example the vehicle name and vehicle type.

The actual measurement collection happens during collection rounds. During one round measurements for a time period are fetched from all datanodes of each site, converted to entity updates, and sent to FIWARE. Then another round is started with the next time period. The length of the time period varies. During real time collection the period is one minute and there may be a short waiting period before the next round to ensure that new measurements have arrived to IoT-Ticket. During measurement history collection the time period is two hours.

During a collection round the following steps are performed:

1. Measurement values for all datanodes of each site are fetched for the time period with the IoT-Ticket API endpoint that returns measurements for a datanode for the given time range.
2. For each measurement value convert timestamp from microseconds to seconds, round to second precision, and ensure that we have at most one value for a datanode for a single second. See 4.3 for reasons why.

3. For each site go through latitude and longitude values and combine those with the same timestamp into a suitable value for a FIWARE location attribute.

4. For each site build entity updates from datanode values so that each update has attributes whose values share the same timestamp. See 4.3 for explanation.

5. For each entity sent its updates to Quantumleap via its notify API with max 600 entity updates per request. See 4.4 for why Orion subscriptions are not used.

6. For each entity build an update that has the newest value for each attribute and send these to Orion. Note unlike in the Quantumleap updates, with these updates the attributes do not have to share the same timestamp i.e. we do not have to take the varying measurement update intervals into account.[1]

4 Experiences and lessons

This section describes experiences and lessons learned during the development of the data model and tool described in the previous section.

4.1 The data model and data conversion

Making the data model was quite simple since we only needed one entity type for the bus and we could adapt an existing data model for it. Data conversion was also quite simple because the IoT-Ticket data model also matched nicely to the FIWARE data model in this case. We could just map the bus sites into Vehicle entities and match each datanode to an entity attribute. The only exception being the latitude and longitude datanodes which were combined into the standard FIWARE location attribute. Another small oddity was that the name of the datanode containing the speed measurement was not the same with all buses. This demonstrates, that data can always contain exceptions, so some care is required when converting data.

We wanted to make the data model human friendly so some extra effort was required. Some datanodes, which did not have a corresponding existing attribute and thus required a new entity attribute, had long or odd names. In these cases we did not just give the new attribute the same name as the datanode but instead created a new one. For example datanode named *battery state of charge* was mapped to attribute *batteryState*. Some datanode values were not obvious so in the data model they were given more human friendly values. For example door status was indicated with numeric values 0, 1 and 2 which were mapped to textual values open, closing and closed. Similarly datanode *vehicle*

[1] This step is optional and can be omitted when we are collecting historical measurements but Orion already has newer measurements collected previously.

motion had values 0 and 1 which were mapped to boolean attribute *isMoving*. During the development of the tool we noticed that these measurements with an enumerated set of possible values can also have other values we assumed due to an error with the measurement system. We decided just to ignore these values but this shows that you may get unexpected data and you should be prepared to handle it.

Making this kind of more human friendly data model requires deeper understanding of the source data and if required as in our case communication with people who are more familiar with the data. In addition to above mentioned issues we also required explanations for the meaning of some datanodes since they were not apparent from the name alone. We also needed the actual names of the buses since in IoT-Ticket they were just named bus, bus1, bus2 etc.

4.2 APIs

The API of the data source, IoT-Ticket in this case, affects how efficiently data can be collected and what way of collecting is the best depending on the goals. In this case the biggest restriction was that there was no API endpoint for getting measurement history for multiple data nodes with one request. Thus in order to get all measurements for a time period we have to make a request for each datanode. Together these requests made one after one takes about 20 - 30 seconds. Making the requests concurrently might make the operation more efficient but might cause performance issues with IoT-Ticket.

With the current way of collecting the data updating of Orion happens every minute with a 60 second delay from current time. This way of collecting may not be suitable for applications that would benefit from more real time measurements. This kind of collection could be implemented since IoT-Ticket offers another endpoint that can give the most recent measurements for all datanodes of a site. By using this API endpoint we could have implemented a feature that continuously gets the most recent values from IoT-Ticket and sends them to Orion while at the same time the existing round based collection would still send measurements to Quantumleap. This would be required since by always getting only the most recent values we could not ensure that we get all values available. This kind of collection might however cause performance issues with IoT-Ticket.

4.3 Timestamps

The way FIWARE data models can combine multiple measurements, that can have their own timestamp in the original data source, under one entity attribute with a single timestamp can cause issues when converting measurements. Such a case was with the latitude and longitude datanodes and the FIWARE location attribute. Each datanode has its own timestamp so we had to combine values from the latitude and longitude datanodes under a single timestamp. In most cases there were a latitude and longitude value with the exact same timestamp

but for some reason sometimes a value was missing so this had to be taken into account when converting the data.

The way Quantumleap stores measurements and based on that represents them to users causes issues with measurements belonging to the same entity but updated independently from each other. IoT-Ticket records timestamps in microsecond precision and the bus measurements each have their own timestamp which mostly are unique. Quantumleap on the other hand expects that all attributes included in one entity update share the same timestamp. It also stores attribute values with the same timestamp in one database row and this is also reflected in the returned JSON when requesting measurement values with the Quantumleap API. This is why we round measurement timestamps to second precision and combine attributes sharing the same timestamp to updates in the data collector. Without rounding the results for getting values for multiple attributes would look like data in the table 1. There each row has just one measurement value where as with rounded timestamps one row would have multiple values as shown in the next section's table 3.

Table 1: Data in Quantumleap when timestamps are not rounded.

Timestamp	airTemperature	chargeState	doorStatus	energyConsumed
2019-05-27T08:59:59.562	null	77.0	null	null
2019-05-27T09:00:00.393	null	null	closed	null
2019-05-27T09:00:01.009	null	null	null	9.6
2019-05-27T09:00:01.421	10.90625	null	null	null
2019-05-27T09:00:03.422	10.90625	null	null	null
2019-05-27T09:00:04.661	null	77.0	null	null
2019-05-27T09:00:05.478	null	null	closed	null

4.4 FIWARE Orion subscriptions

The conventional FIWARE approach for sending the data to Quantumleap or a similar history data service would be just to create suitable Orion subscriptions and then only send the data to Orion . The subscription system would then take care of sending the data to Quantumleap. However in this case we encountered issues related to correct handling of timestamps and performance. Thus in the end we had to sent the measurements directly to Quantumleap.

The previously described independently updated attributes seem to be challenging for the subscription system and Quantumleap. We explored two ways of making the subscription. In the first way we created a simple subscription where if any attribute changes all current values are sent to Quantumleap. The issue with this subscription is that we lose the original update time for each measurement as shown in the table 2. It gives the impression that we have a value for each measurement for every second when this is not the case as shown by table 3 which shows the data when measurements are sent directly to Quantumleap. It shows for example that we have only two measurements for

the door status and battery charge state. Recently a new option was added to Orion subscriptions which sends only those attribute values that have changed. It might have fixed this issue but we have not tested it. In the second way of making subscriptions we created separate subscriptions for each attribute so that when the attribute is updated only its value is sent to Quantumleap. The issue with these subscriptions was that we got duplicate entries for the same timestamp with a single attribute value as demonstrated by table 4.

Table 2: Data in Quantumleap when using a simple subscription.

Timestamp	airTemperature	chargeState	doorStatus	power
2019-05-27T09:00:01.000	10.90625	77.0	closed	-34.8
2019-05-27T09:00:02.000	10.90625	77.0	closed	-54.2
2019-05-27T09:00:03.000	10.90625	77.0	closed	-83.4
2019-05-27T09:00:04.000	10.90625	77.0	closed	-103.9
2019-05-27T09:00:05.000	10.90625	77.0	closed	-150.0

Table 3: Data when send directly to Quantumleap.

Timestamp	airTemperature	chargeState	doorStatus	power
2019-05-27T09:00:01.000	10.90625	null	null	-34.8
2019-05-27T09:00:02.000	null	null	null	-54.2
2019-05-27T09:00:03.000	10.90625	null	null	-83.4
2019-05-27T09:00:04.000	null	null	null	-103.9
2019-05-27T09:00:05.000	null	77.0	closed	-152.0

Table 4: Data in Quantumleap when using attribute subscriptions.

Timestamp	airTemperature	chargeState	doorStatus	power
2019-05-27T09:00:01.000	10.90625	null	null	null
2019-05-27T09:00:01.000	null	null	null	-34.8
2019-05-27T09:00:02.000	null	null	null	-54.2
2019-05-27T09:00:03.000	null	null	null	-83.4
2019-05-27T09:00:03.000	10.90625	null	null	null
2019-05-27T09:00:04.000	null	null	null	-103.9
2019-05-27T09:00:05.000	null	77.0	null	null
2019-05-27T09:00:05.000	null	null	closed	null
2019-05-27T09:00:05.000	null	null	null	-152.0

Another issue with the subscription system was performance. With both types of subscriptions not all data arrived to Quantumleap with even third of the data going missing. Do ensure that all data arrived to Quantumleap we had to slow down the data sending by waiting between requests to Orion. Partly this issue is due to the inefficient way Orion sends notifications with a single notification containing only one entity update. Orion could be smarter and attempt to gather multiple updates to a notification.

4.5 Fault tolerance

Since the collector runs continuously when collecting real time data or a long time when collecting historical data, and there can be issues with both receiving the data and sending it, it was important to make the tool tolerate these kind of issues. When making HTTP requests we simply repeat a failing request after 10 seconds. When real time collecting and there is a pause in collecting due to an error we will catch up to the current time by extending the time period measurements are collected from during a collection round.

Sending the measurements directly to Quantumleap instead of using the subscription system allows also better fault tolerance. Even aside previously described performance issues there is no way to ensure that a subscription notification has come through to Quantumleap, for example in case where it is down. Orion can only report the status of newest notification and the time of last successful notification when requested.

5 Conclusions

Although this data conversion and transferance was quite simple from the source data perspective we encountered surprising issues which made this work more complicated than assumed. The main issues were the way timestamps are handled and how the FIWARE subscription system works. This work also shows that even a simple data model requires understanding the source data. Overall this work was a success and it gave us valuable lessons about FIWARE.