Integrating Online Data for Smart City Data Marts *

Michael Scriney[1], Martin F. O'Connor[2], and Mark Roantree[1]

[1] Insight Centre for Data Analytics, School of Computing, Dublin City University, Glasnevin, Dublin 9, Ireland
[2] Department of Computing, Institute of Technology Tallaght, Dublin 24, Ireland
michael.scriney@insight-centre.org, mark.roantree@dcu.ie, martin.oconnor@it-tallaght.ie

Abstract. The development of smart city infrastructures is seen as an important strategic component for many countries with applications in the areas of government services, healthcare, transport and traffic management, energy, water, and the management of waste services. In general, each of these infrastructural components generates data as it delivers its service. The usage of this data is often crucial both to the continued development of the service and for forward planning. While the data sources are often not complex in structure, traditional decision support systems are not built to handle continuous data streams, new or disappearing data sources, or the integration of multiple online data sources. In this work, we develop a flexible ETL process to quickly process and integrate online smart city data sources to deliver information that can be used in close to real-time.

1 Introduction

The Internet of Things has provided a means of collecting and analysing previously inaccessible data across a wide variety of topics. Recently, there has been an emergence of publicly available data regarding the infrastructure of various cities. This data, provided by governmental agencies and private industry can lead to direct benefits for the citizens of a city. The projects which process, store and analyse this data are known as Smart City applications. Data in a smart city is generated from a wide variety of sources across a large number of domains such as housing, environment, transport and so on. Similar to traditional web data, this data is available on-line, typically in either XML, JSON or CSV format and there have been many approaches to building smart city applications [9] and to clustering or integrating query graphs on smart city data [13]. Most of the query or search engines are RDF based eg. [4] but recently, we have seen the emergence of approaches to understanding JSON schemas [1] which allows for scaling of large online datasets.

* This work is supported by Science Foundation Ireland under grant number SFI/12/RC/2289

Decision makers who use smart city applications expect to have some form of OLAP service which provides the datasets from a warehouse upon which they make their decision. However, online data sources are not included in many data warehouses for many reasons. These may include lack of control of the source data, lack of understanding of how to process the data, a constantly changing structure to the data or issues to do with quality control in terms of the data's content. Access to these services generally requires an API supplied by the service provider or some form of wrapper to extract the data. In either case, there is a time lapse between when new online information becomes available and when it is accessible using OLAP tools for decision makers. Consider a scenario where emergency services require the fastest route between two points in the city: unless data is available in close to real-time it is of little use to the route planning service. In this paper, we will present an approach to making online data available in smart city data marts in close to real time. To deliver this service, we update a Data Lake every 30 seconds from source data. Briefly, a Data Lake is a data store which stores data in its original native format. By close to real time, we mean that online information providers are checked every 30 seconds for updates, those updates (if any) are populated into data marts that use our Star Graph format [15] to provide data immediately to decision makers.

1.1 Background and Problem Statement

In order to deliver integrated datasets from multiple external sources, what is required is a sufficiently robust Extract-Transform-Load (ETL) process so that facts and dimensions can easily be identified; external sources are mapped to some form of ontology, and an integration process to merge the specific data sources that are needed for specific tasks. An approach advocated in [2] is to use lightweight dynamic semantics which our system uses to maintain interoperable descriptions of data sources inside the Data Lake. This metadata is then used to update a data mart in close to real-time. In our approach, we do not assume that a predefined schema and ETL process exists. This is traditional ETL where the warehouse schema is designed upfront, and data is extracted from all (known) sources in advance, integrated according to strict rules and data is loaded whether it is required or not.

The problem with online data sources is that they evolve much more than in-house databases. Sources can disappear or change structure; new sources can become available; different user needs often require the construction of heterogeneous data marts. The problem with applications such as smart cities are that in order to exploit the wealth of information available, it requires a more flexible approach to ETL and data mart construction. At the same time, we aim to provide OLAP style functionality to create the datasets necessary for analysis of long duration data or small data marts with only the most recent updates. This solution must also take into consideration that data marts should be dropped where data sources are no longer available or newly created where new sources

have become available as data marts may use different combinations of source data.

Contribution. In earlier work [15], we presented a system to automatically create multi-dimensional data marts from online sources. We employ a graph representation for data marts which is called a *StarGraph*, a graph structure which comprises the multi-dimensional concepts of *facts* and *dimensions*. In this work, we present a mechanism for integrating StarGraphs. While the StarGraph has a 1-1 relationship with external sources, a second structure, the *ConstellationGraph* is an integration of StarGraphs, with 1-many mappings with the StarGraphs which comprise it. In addition, we can operate real-time data marts where data is pulled from the Data Lake containing online sources on demand.

Paper Structure. The paper is structured as follows: in §2, we present related research; in §3, our case study is introduced; in §4, we describe the processes involved in our system; in §5, we provide an evaluation of our work; and finally, in §6, we provide some conclusions.

2 Related Research

Similar to our research, the authors in [16] use graph structures to model heterogeneous data sources which are to be integrated and stored in a data warehouse. However, the authors employ the use of application designers to manually add an ontology to each graph structure in order to produce mappings from source to target schemas, whereas our process seeks to automatically identity facts, dimensions and measures and provide a corresponding mappings file for further analysis of data sources.

In [12], the authors present a means of providing multidimensional analysis through the use of a multidimensional ontology. This ontology itself is built from a semantic data warehouse, which provides ontologies about all source data required for multidimensional analysis. However, this does not provide the flexible and more lightweight approach [2] necessary for online data sources. In [3], the authors again present a smart city ontology which is combined with a data collection system to harvest smart city data sources to provide smart city applications. Similarly to our paper, the authors focus on transport data. However, while their system utilises an ontology in order to provide mappings and integration, our system is more flexible in that it automatically generates mapping files and identifies facts and dimensions for further analysis.

In [11] the authors present a real time smart city application which tracks and monitors city facility utilisation. The authors use a system which provides OLAP functionality on a data stream. However, the integration methodology for various data sources used by the authors is a manual process, which requires a custom wrapper to be developed per data source, whereas our system provides both a manual and automatic means of integrating data.

In [8] the authors detail a smart city system in Barcelona. The system is composed of a number of sensors available through HTTP/REST interfaces. The authors note there is a difficulty in the management of the data due to

the large volume generated by multiple sensors. The smart-city sensors generate data in the form of JSON which is stored in a database made available to third-parties over XML. Similar to our integration approach, the authors note that an intelligence module is provided which correlates data in space and time, the authors do not however detail how this system works, neither do they detail how the data is integrated and stored in the central database.

3 Case Study Introduction

We will now introduce our case study which will be used to conceptualise the need for a StarGraph and provide a means of illustrating the processes involved throughout the lifecycle of our system. Commuters use various means of transport in order to reach their place of work. However, due to the ever increasing number of people who commute the transport infrastructure can become strained. This leads to increased commuting times. There are benefits to both the transport providers and the commuters themselves for minimising travel times across the city.

The transport infrastructure of a city can be conceptually viewed as a graph. A node on the graph indicates a point of departure or a destination (for example, a train station, a bus stop, a road junction and so on). An edge between two nodes indicates a path. A contiguous sequence of two or more paths is a route. This topological graph may be used to find a path for the commuter between their departure and destination with the shortest distance. However, when commuting, distance covered is not the only feature of interest. For many commuters, the minimum time taken to commute is preferable to distance covered.

In order to determine the most efficient route at time t a collection of historical transport data must be collected. To satisfy this case study, the following functionality must be provided:

1. Harvesting of multiple domain-related data sources.
2. Integration and contextual enrichment of the data.
3. Population of the integrated data warehouse.
4. Provision of a Query Service.

A number of Ireland's transport providers publish real-time data, available through the use of public APIs. For the purposes of this case study the following datasets have been chosen.

- Dublin bus RTPI [6], details the next buses due at a specified bus stop.
- Dublin Bikes [5], provides real time indication of bike availability.
- LUAS (Tram line) [17], Details the next arrivals along a tram line.
- Motorway travel times [18], The travel time taken to traverse the motorways.
- Dublin City Travel Times [7], Time taken to traverse junctions on Dublin Roads.
- Irish Rail API [10], Details the upcoming arrivals at a given train station.

In this paper we will demonstrate how our system can be used to find the most time-effective means of getting a commuter to their destination.

4 Process Framework

There are four main processes to our system as shown in Fig. 1.

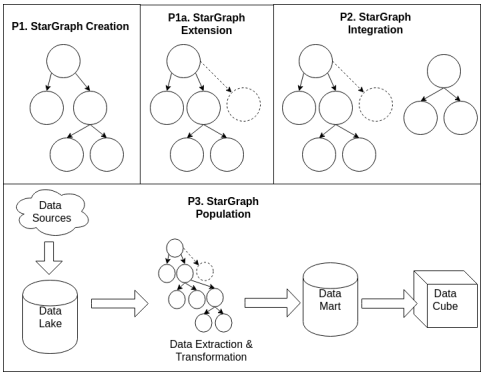

Fig. 1: A StarGraph-based ETL System

1. **P1. StarGraph Creation**. The first process in our system creates a StarGraph from the source schema.
2. **P1a. StarGraph Transformation**. This is an optional step which can be used to enhance the structure of the StarGraph by allowing a user to manually edit the graph, or add supplementary information necessary for further analysis.
3. **P2. StarGraph integration**. This process takes one of two forms: integration of 2 StarGraphs to form a ConstellationGraph; or the integration of a StarGraph and ConstellationGraph which allows continued growth of the final schema.
4. **P3. StarGraph Population**. This process populates a fact table based on the mapping rules. It has two inputs: the mapping rules corresponding to the data mart (P3) and the data stream from which the data is to be extracted.

4.1 StarGraph Creation (P1)

A StarGraph represents the facts, dimensions and measures found inside a data source. In addition, it provides a mappings file which can be used to transform the source data into the StarGraph representation. For the case study presented here, when the last source file is updated, the StarGraph is then updated from all sources. For the 6-source example shown in Table 1, updates occur between 30 and 300 seconds.

Table 1 details the StarGraphs created from the data sources detailed in §3 along with the time taken to populate from the data streams and the number of new instances constructed for 1 sample update.

Table 1: Constructed StarGraphs

Name	ID	Format	Nodes	Edges	Dimensions	Measures	Population(ms)	Rows
Dublin Bikes	bik	JSON	14	14	2	4	69	245
Dublin Bus	bus	JSON	23	23	2	0	24	101
LUAS	lus	HTML	3	3	1	0	7	2
City Roads	rds	CSV	7	0	1	7	64	902
Motorway	mot	JSON	18	2	10	3	14	25
Irish Rail	ril	XML	22	23	1	2	10	2

Generating Mappings. In addition to the multidimensional representation produced by the StarGraph process, a corresponding series of mapping rules are produced to facilitate the data extraction and transformation from source to target. When a schema is first read, a list of all possible paths through the schema and their corresponding points in the StarGraph is produced. This metadata is used as a basis when formulating the Mapping Rules for the StarGraph. Mappings are stored in JSON format.

Mappings are only generated once the fact table has been constructed from the StarGraph. The mappings database contains:

- Dimension - Indicates a full dimension object.
- name - Indicates the name of either an attribute or measure.
- src -This attribute details the location of the relevant data in the data stream. It can take the form of an xpath query, JSON dot notation or a function definition (§4.2).
- id - Id denotes the primary key of a table. The id attribute can only be found inside an Dimension object or Subdimension. This can either be automatically generated (in the absence of a defined key) or is created using a src query.
- table - The name of the table to store the data
- atts - Atts is a list of attributes associated with a dimension or subdimension
- type - Type is the data type of the attribute.
- subdimensions - This is a list of subdimension objects.
- measures - These are a list of measures which are found.

4.2 Enrichment & Transformation (P1a)

Data Transformation For some datasets, the measure may not immediately be detected. This issue can arise when a transformation function is necessary to expose the measure. For example, the Dublin Bus dataset does not provide a unique measure (i.e. when the next bus is due) but rather provides two timestamps one named `scheduledarrivaldatetime` which refers to the time the bus is expected at a stop, and another named `timestamp` which refers to the current timestamp. A useful measure would be when the next bus is due in terms of seconds (or minutes). With a created StarGraph and mappings file, this file can be extended by an application developer to provide a transformation function (Fig. 2).

```
{"name":"timeInSeconds",
 "src":function(){ return
 $scheduledarrivaltime.getTime()- $timestamp.getTime(); },
 "type":"int"}
```

Fig. 2: Example of user defined transformation

By placing this function under the `Measures` section of the mappings file, the code will be executed. This result will then be stored in the fact table under the name `timeInSeconds`. If it is placed in the `Dimension` or `Subdimension` areas of the mappings file, it will be stored as an attribute of the Dimension.

At present, the mapping files are stored as JSON, and transformation functions are written in JavaScript. When new data is introduced to the system, the mappings file extracts all the source data for population. In addition, these transformation functions are evaluated in order to produce new measures for analysis as the system is being populated.

Providing Streaming Context Many smart city data streams provide data in a format coded by an application designer, and not by a business user. An example of data coding would be providing unique id's to objects. However, oftentimes supplementary data relevant to the code is not provided as part of the data stream, but as a static file. These static files can contain geolocation data, fully resolved addresses etc. which are tied to the unique ids provided by the stream. Static file integration seeks to resolve these differences by re-combining the static supplementary data to the data stream.

An example of static file integration can be seen if we examine the City Roads dataset. The City Roads dataset is a CSV file which provides little information about the data being accessed (Fig. 3). This is very similar to body sensor applications which produce very high volumes of information [14] but where the data generated is very simplistic and requires external semantics to increase its impact.

#Route	Link	Direction	STT	AccSTT	TCS1	TCS2
1	1	1	128	128	2127	175

Fig. 3: Example of the CityRoads Dataset

In this data, the headers `TCS1` and `TCS2` refer to real world locations. However, without the supplementary descriptive data, the numeric data lacks meaning. The data provider (Dublin City Council) also provides a KML file (geographic annotation) (Fig. 4) which provides the additional data necessary to understand the real time data. This supplementary data can be used to enrich the existing CSV file with latitudes and longitudes linking the data TCS1 and TCS2. The supplementary data is supplied by the same provider as the stream, as such there is a one-to-one mapping between the terminology used for the stream and the static file, therefore, this data can be integrated using a simple text matching operation which can be used to link the `coordinates` tag to the existing data.

Additionally the mappings file associated with the StarGraph has been extended to map the data stream to the provided context.

```
<SimpleData name="TCS1">6006</SimpleData>
<SimpleData name="TCS2">2031</SimpleData>
        <coordinates>-6.172557196923532,53.291529410712464
      -6.184390631942438,53.296472990543961</coordinates>
```

Fig. 4: KML file (truncated) provided by Dublin City Council

4.3 StarGraph Integration (P2)

Two or more StarGraphs are integrated to produce a `Constellation Graph` with a process that can be either manual or automatic. For manual integration, a user selects an integration attribute for both StarGraphs and they are joined on that attribute. Once the StarGraphs have been joined, changes to dimensions and measures require the generation of new mappings. These mappings are similar to the mappings described in §4.1. They are generated in a similar fashion to that of a StarGraph, except in this instance an attribute may have multiple `src` properties. Automatic integration combines StarGraphs using an ontology and matching attributes based on name, location or date. For example Figures 5 and 6 represent two StarGraphs which are to be merged. The nodes `datetime` from the Bikes StarGraph and `timestamp` from the Bus StarGraph are merged because they occupy the same date dimension.

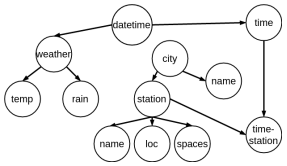

Fig. 5: Bikes StarGraph

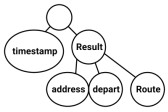

Fig. 6: Dublin Bus StarGraph

4.4 Populating and Updating (P3)

As indicated earlier, the trigger for updating a ConstellationGraph is when the last source file has been updated. The mappings database is used to transfer data from the Data Lake into the warehouse (StarGraph).

Querying. Once a StarGraph (or ConstellationGraph) has been populated, the location data found inside the StarGraph can be used to construct a topology representing the physical distribution of the data. If there are multiple latitudes and longitudes occupying the same instance, an edge is created between them.

Recall the inputs for the case study are a points of departure and destinations and the datetime of departure. The first step in completing any valid query is to examine the topological graph to determine all possible paths which lead from

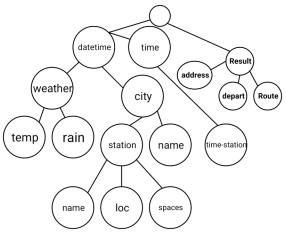

Fig. 7: ConstellationGraph constructed from the bikes and bus StarGraphs

the departure point to the destination. For the purposes of this case study this is achieved using a depth first search to determine all simple paths. Once all simple paths have been constructed, these are stored as a series of possible routes.

The next step is to examine which route is the fastest for the departure time the user provided. The historical aggregate of the travel time for each route at time t is evaluated and the minimum route is returned.

5 Evaluation

The goal of our evaluation was to measure how close to real-time, we can deliver the datasets necessary to compute information for travel requirements. The travel case study has six data sources and we used 57 different ConstellationGraph (schema) configurations to measure the usability of the system and potential scalability. Table 2 reports from a representative batch of 12 of these configurations. In each case, we report the time taken in milliseconds to refresh each schema when an update is detected. An update in this case is where change was detected in *all* data sources with the trigger occurring when the final data source has changed. Our experiments were completed using Node v6.10.0 64-bit, Mongo v3.4.2 64-bit on a SONY VAIO with 12GB RAM and Intel i7-3632QM 2.20GHz, Windows 10 Home edition 64-bit

Datasets were separated into large and small, based on the number of new instances, with `bus`, `bik` and `roads` constituting the large dataset, and the remainder making up the small one. We are not loading large data volumes as updates occur no longer than 5 minutes apart: the goal is to make current data available.

Column `2a` was created from combining two sources from the large datasets (specifically, the `bus` and `bik` datasets). `2b` was created from combining one large and one small dataset (`bus` and `lus`). This update generated 270 new instances and was completed in 76ms. Interestingly, for population of two data sources, `2c` which was created by combining two small data sources (`lus` and `ril`) took the largest time of 157ms which resulted in a fact table consisting of only 4 rows. This indicates that it is not the number of rows generated that impacts more on time but rather the format of the data sources itself. As these sources are HTML and XML respectively, the time taken to parse these schemas compared their JSON and CSV counterparts has a large impact on the integration times.

Columns 3a, 3b and 3c show the times taken to populate a schema (data mart) composed of three sources. 3a was created from all three of the large data sources (bus, bik and rds), 3b was created from two large and one small data source (bus, bik and mot) and 3c was created from all of the small data sources (lus, mot, ril). Once again, the time taken to populate a fact table composed of all the large data sources, and the small ones were comparable (156 and 159ms respectively). These times were despite the fact that the large data sources result have 1219 more instances than small sources.

Columns 4a , 4b and 4c show the times taken to construct a schema from four data sources, with 4a being created from 3 large datasets, and one small; 4b was created from three small datasets and 1 large; and 4c was created from two small and two large datasets. The times taken to populate each fact table were 173ms, 210ms and 188ms respectively, with 4b (bik,lus,mot and ril) taking the largest time. This fact table was constructed from 2 JSON sources, 1 XML source and 1 HTML source while 4c (which was constructed from 2 JSON files, 1 CSV file and 1 XML file) proved to be faster by 22ms. Once again, the time difference between 4b and 4c indicates that the time taken to populate a data mart is largely due to the format of the source data streams. We can see this if we compare the times taken for both 4b and 3b as they both used the same data sources (apart from the HTML source lus. With 3b being populated in 88ms, and the addition of a fourth source lus as shown in 4b increases the population time by 122ms.

Columns 5a and 5b show the times taken and the number of rows constructed for schemas created from 5 data sources, with 5a being created from 3 large datasets and 2 small ones and 5b being constructed from 3 small datasets and 2 large ones. Interestingly at this point, it appears that the number of instances created appears to surpass the initial bottlenecks provided by the source data formats, with 5a being populated in 310ms and 5b in 254ms. In short, as the number of data sources increases, the largest factor on population time, changes from the formats of the source schemas, to the number of rows produced by the population.

Finally, column 6 shows the time taken to populate the schema created from all 6 sources. The population time for all sources was 350ms and resulted in a a fact table of 1277 rows. Undoubtedly this configuration yielded the largest results for both population time and rows produced, when compared to 5a which is made from all data sources apart from ril the difference in time between the two is 40ms, while the differing number of rows is only 2 rows. This indicates that, as the number of data sources increases, the number of rows new instances has a noticeable impact on performance.

With respect to the individual data sources presented. The inherent complexity of each schema does not seem to pose a problem during population, most likely due to the fact that the mappings file details the exact location of the data in each stream which is required for population. Overall, the time taken to populate a fact table from combined data sources is not so much limited by the number of individual facts (rows) per data source, but rather the format and

structure of the source data provided. With formats such as XML and HTML taking a longer time to process than those which are published as JSON or CSV files. However, it appears that as the number of sources involved in the fact table population increases, the resultant number of instances to be created becomes the dominant factor in population time rather than the format of the data stream.

In summary, our system integrates and makes available in close to real-time the data produced by the different transport services, and experiments show a linear rise in times with an increase in the size of the update. The evaluation also demonstrated that:

- XML and HTML sources (bold font in table 2 increase the cost of updates (157ms for only 4 new instances in 2c)
- The HTML dataset (lus) performed notably worse than XML showing that more structured data performs better (5a *vs* 5b).
- Larger data marts with more data sources have less of an effect than an increase in new data instances (3b *vs* 2c).

Table 2: Update Times for Various Data Mart configurations

Config ID	2a	2b	2c	3a	3b	3c	4a	4b	4c	5a	5b	6
Souces	bus bik	bus **lus**	**lus** **ril**	bus bik rds	bus bik mot	**lus** mot **ril**	bus bik rds mot	bik **lus** mot **ril**	bus mot rds **ril**	bus bik **lus** mot rds	bus **lus** mot rds **ril**	bus bik **lus** mot rds **ril**
Population (ms)	91	76	157	156	88	159	173	210	188	310	254	350
Rows	346	270	4	1248	371	29	1273	130	1174	1275	1176	1277

6 Conclusions

In this paper, we demonstrated how StarGraphs can be used to create the ConstellationGraph which serves as a Data Mart in decision support systems. In addition, we provide a fast method for incorporating new data sources as they come online and our evaluation shows that these sources are in our data marts (almost) immediately after update. Our future work involves a project that uses 200 Agri (agricultural) data sources to provide more of a scalable stress test for this system. Secondly, we are investigating how the system can suggest to the user which combinations of datasets should be incorporated into the same data mart.

References

1. Baazizi, M.A., Lahmar, H.B., Colazzo, D., Ghelli, G., Sartiani, C.: Schema inference for massive json datasets. In: Extending Database Technology (EDBT) (2017)

2. Barnaghi, P.M., Bermúdez-Edo, M., Tönjes, R.: Challenges for quality of data in smart cities. J. Data and Information Quality 6(2-3), 6:1–6:4 (2015)
3. Bellini, P., Benigni, M., Billero, R., Nesi, P., Rauch, N.: Km4city ontology building vs data harvesting and cleaning for smart-city services. Journal of Visual Languages & Computing (2014)
4. Cappellari, P., De Virgilio, R., Maccioni, A., Roantree, M.: A path-oriented rdf index for keyword search query processing. In: Database and Expert Systems Applications. pp. 366–380. Springer (2011)
5. Dublin Bikes API [Online] (originally available in XML, now only available in JSON): `http://api.citybik.es/v2/networks/dublinbikes`
6. Dublin Bus RTPI [Online]: `http://dublinked.ie/real-time-passenger-information-rtpi/`
7. Dublin City travel times [Online]: `http://opendata.dublincity.ie/TrafficOpenData/CP_TR/trips.csv`
8. Gea, T., Paradells, J., Lamarca, M., Roldan, D.: Smart cities as an application of internet of things: Experiences and lessons learnt in barcelona. In: Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference On. pp. 552–557. IEEE (2013)
9. Hernández-Muñoz, J.M., Vercher, J.B., Muñoz, L., Galache, J.A., Presser, M., Gómez, L.A.H., Pettersson, J.: Smart cities at the forefront of the future internet. Springer (2011)
10. Irish Rail [Online]: `http://api.irishrail.ie/realtime/index.htm?realtime_irishrail`
11. Komamizu, T., Amagasa, T., Shaikh, S.A., Shiokawa, H., Kitagawa, H.: Towards real-time analysis of smart city data: A case study on city facility utilizations. In: HPCC/SmartCity/DSS, 2016 IEEE 18th International Conference. pp. 1357–1364. IEEE (2016)
12. Nebot, V., Berlanga, R., Pérez, J.M., Aramburu, M.J., Pedersen, T.B.: Multidimensional integrated ontologies: A framework for designing semantic data warehouses. In: Journal on Data Semantics XIII, pp. 1–36. Springer (2009)
13. Roantree, M., Liu, J.: A heuristic approach to selecting views for materialization. Software: Practice and Experience 44(10), 1157–1179 (2014)
14. Roantree, M., McCann, D., Moyna, N.: Integrating sensor streams in phealth networks. In: Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on. pp. 320–327. IEEE (2008)
15. Scriney, M., O'Connor, M.F., Roantree, M.: Generating cubes from smart city web data. In: Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2017, Australia, 2017. pp. 49:1–49:8 (2017)
16. Skoutas, D., Simitsis, A.: Ontology-based conceptual design of etl processes for both structured and semi-structured data. International Journal on Semantic Web and Information Systems (IJSWIS) 3(4), 1–24 (2007)
17. Transport for Ireland [Online]: `http://www.transportforireland.ie/real-time/real-time-ireland/`
18. Transport Infrastructure Ireland [Online]: `https://www.tiitraffic.ie/travel_times/`