12th ITS European Congress, Strasbourg, France, 19-22 June 2017

Paper ID #

Constraints for a large-scale ITS data-sharing system: a use case in the city of Ghent

Pieter Colpaert[1*], Peter Van der Perre[2], Dieter De Paepe[1], Thimo Thoeye[3], Ruben Verborgh[1], Erik Mannens[1]

* pieter.colpaert@ugent.be

1. IMEC – Ghent University – IDLab

2. ITS.be

3. City of Ghent

Abstract

Thanks to architectural constraints adopted by its stakeholders, the World Wide Web was able to scale up to its current size. To realize the ITS directive, which stimulates sharing data on large scale between different parties across Europe, a large-scale information system is needed as well. We discuss three constraints which lie at the basis of the success of the Web, and apply these to transport data publishing: stateless interaction, cacheability and a uniform interface. The city of Ghent implemented these constraints for publishing the dynamic capacity of the parking sites. The information system, allowing federated queries from the browser, achieves a good user perceived performance, a good network efficiency, achieves a scalable server infrastructure, and enables a simple to reuse dataset. To persist such a transport data information system, still a well-maintained Linked Data vocabulary is needed. We propose to add these URIs into the DATEX2 specification.

Keywords:

Linked Data, Information Architecture, Parking Sites

Introduction

The data-driven smart city is only smart to the extent the city's datasets can be discovered and used by the *user agents* of its citizens. Instantly, an organizational problem arises, to structure all the city's data for world wide adoption. Opening this data would, for instance, allow a user agent to automatically detect the underground parking sites when a car approaches a city of choice, their availability, opening hours, extra facilities, vicinity to public transport, public transport time tables or points of interest.

Today, data sharing for maximum reuse – *Open Data* – is slowly emerging on a pan European scale as a policy framework is put into place. The European Public Sector Information (PSI) directive laid the foundation of a data publishing policy across different members states. The directive was the start for

Public Sector Bodies across Europe to, both passively as pro-actively, share data on the Web using national Open Data portals. As the implementation details of the directive however are still left open to interpretation, this legal obligation would result in member states to create their own strategy for enabling Open Data [5]. Furthermore, within the domain of Intelligent Transport Systems (ITS), the ITS directive helped popularizing publicly sharing data. For example, with a delegated regulation elaborating on a European Access Point for Truck Parking Data [4], it regulates sharing the data through a national access point similar to the INSPIRE directive, a directive for sharing data within the geo-spatial domain.

When creating a system to distribute data within our own domain – for the many years to come – an important requirement is that this data policy needs to *scale up efficiently*. When more datasets are added, when more difficult questions need to be answered, when more questions are asked or when more user agents come into action, we want our system to work without architectural changes. Furthermore, the system should be able to evolve while maintaining *backwards-compatibility*, as our organization is always changing. Datasets that are published today, still have to work when accessed when new personnel is in place. Such a system should also have a *low entry-barrier*, as it needs to be adopted by both developers of user agents as data publishers. In this paper, we summarize the literature from the Web as a scalable Information Management System and apply the insights to ITS.

A large-scale information system

What we now call "The Web" [2] is a knowledge base with mankind's data, which still uses the same building blocks as at the time of Tim Berners-Lee's first experiments. It was Roy Fielding who, in 2000 – 11 years after the initial proposal for the Web – derived a set of constraints to explain the Web's properties. Defined while standardizing HTTP/1.1, this set of "how to build large knowledge bases" constraints is known today as Representational State Transfer (REST). For a comprehensive overview of REST in its entirety, we refer to a review of REST after 15 years [2], or the original dissertation of Roy Fielding (2000) [3]. As with any architectural style, developers can choose to follow these constraints, or to ignore them. When following these contraints, REST promises beneficial properties to a system, such as a good user perceived performance, network efficiency, scalability, reliability and simplicity for data consumers.

HTTP, the protocol powering the Web, allows actions to be performed on a given resource, called request methods or also called HTTP verbs. It is up to data publishers to implement these methods. An example of such method is GET. By executing a GET request, you can download the representation of a certain resource. The same representation can be requested over and over again with the same result, until the resource changes its state – e.g., when a parking site's capacity is reached (a change in the real world), or when someone adds a comment to an article using a POST request –. Furthermore, the protocol specifies a GET request cannot change anything on the server, as it is a "safe" method. As a result, the response to such a request can be cached within clients, in servers, or in intermediate caches.

POST requests are not “safe”. Each time a POST request is executed, a change or side-effect may happen on the server. It can be used to, for example, place a comment that should be saved on the server. It is thus – by definition – not cacheable, as each new request must be able to trigger a new change or must be able to result in a different response. Safe methods can be exploited for maximizing reuse and optimizing cacheability. Non-safe methods are used for ensuring each question reaches the server, and thus, explicitly, cannot be cached.

The HTTP protocol has also more than a handful of other methods, has headers to indicate specifics such as which encoding, negotiating a content-type, how long the response can be cached, and has response codes to indicate whether the request was succesful. Its interface is simple – as only a couple of methods and headers are defined –, powerful and widely adopted.

Stateless interaction

When you are browsing for parking sites in a city and the server has just sent you parking number 23, your client cannot simply say next to the server. Instead, your client has to ask for parking 24 right away. The server does not remember that you were viewing parking 23, and thus your client has to supply all information necessary to execute the request. The good thing is, you don’t have to know either that you were viewing this particular parking: along with the representation of this parking, the server can send you *links* labeled “previous” and “next”, leading to the corresponding parking sites.

Each request from a user agent to a data publisher *must* contain all of the information necessary to understand the request, and a server cannot take into account previous stored information. This is REST’s *stateless* constraint, which requires that each client-server interaction is stateless in nature. The *application’s state* is kept by the user agent based on the links and other controls it followed until now. This design trade-off has a clear draw-back, as more information needs to be sent with every request, even when it was sent earlier. For the use case of data publishing, this does not weigh up to the advantages that the server becomes more scalable as each user agent’s state does not have to be kept in memory, the server’s functionality becomes more easily discoverable as it can be documented with the same uniform interface, and the information system as a whole becomes more r eliable as the user-agent is not lost when one step breaks.

A uniform interface

REST’s *uniform interface* constraint requires that every individual information resource on the Web is accessed through a single identifier – a Uniform Resource Identifiers (URI) – regardless of the concrete format it is represented in. Through a process called content negotiation, a client and a server agree on the best representation. For example, when a resource “parking site 1 in Ghent” is identified by the URI https://stad.gent/id/parking/P1 and a Web browser sends an HTTP request with this URI, the server typically sends an HTML representation of this resource. In contrast, an automated route planning user agent will usually ask and receive a JSON representation of the same resource using the same URI. This makes the identifier https://stad.gent/id/parking/P1 semantically interoperable, since clients consuming different formats can still refer to the same identifier. This identifier is also sustainable (i.e., semantically interoperable over time), because new representation formats can be supported in the future without a change of protocol or identifier. In order to navigate from one representation to another, controls are given within each representation. Fielding called this hypermedia-driven: when a user agent once received a start URI, it would be able to answer its end-user’s questions by using the controls provided each step of the way.

Intelligent agents

We can create a user agent that provides its end-users with the nearest parking site. A user story would look like this: when you push a button, you should see the nearest parking site relative to your current location. In a Service-Oriented Architecture (SOA) – how we would naturally design such an interaction in small-scale architectures – we expose functionality on the server that requires the application to send its current location to the server. A URL of such a GET request could look like this: http://{my-service}/nearestParkingSite?longitude=3.14159&latitude=51.315. The server then responds with a concise and precise answer. This minimizes the data that has to be exchanged when only one question is asked, as only one parking site needs to be transferred. This advantage does not weigh up to the disadvantage…

The number of information resources – or documents – that you potentially have to generate on the server, is too big for an efficient caching strategy. As it is unlikely that two people wanting to know the nearest parking site are at exactly the same locations, each HTTP request has to be sent to the server for evaluation. Rightfully, SOA practitioners introduce rate limiting to this kind of requests to keep the number of requests low. An interesting business model is to sell people who need more requests, a higher rate limit. Yet, did we not want to maximize the reuse of our data, instead of limiting the number of requests possible?

Caching for scalability and user-perceived performance

As there are only 7 parking sites at the city of Ghent, describing this amount of parking sites easily fits into one information resource identified by one URL, for instance, http://linked.open.gent/parking/. When the server does not expose the functionality to filter the parking sites on the basis of geolocation, all user agents that want to solve any question based on the location of parking sites, have to fetch the same resource. This puts the server at ease, as it can prepare the right document once each time the parking sites list is updated.

Despite the fact that now all parking sites had to be transferred to the user agent, and thus consumed more bandwidth, also the user agent can benefit. When a similar question is executed from the same device, the dataset will already be present in the user agent's cache, and now, no data at all will need to be transferred. This raises the user-perceived performance of a user interface. When now the number of end-users increases by a factor of thousand per second – not uncommon on the Web – it becomes easier for the server to keep delivering the same file for those user agents that do not have it in cache already. When it is not in the user agents own cache, it might already be in an intermediate cache on the Web, or in the server's cache, resulting in less in CPU time per user. Caching, another one of the REST constraints, thus has the potential to eliminate some network interactions and server load. When exploited, a better network efficiency, scalability, and user-perceived performance can be achieved.

A data publishing format

Let's take a step back. In order to put data in a document, database or datadump, or in order to transmit a dataset, we need to agree on a serialization first. Examples of common serialization are Comma-Separated Values (CSV), the hierarchical Javascript Object Notation (JSON) or the Extensible Markup Language (XML). The same example dataset is given in three of these serializations in Figure 1.

```
{
 "P1" : {
    "name": "Vrijdagmarkt",
    "parkingSpaces": 648,
    "contact": "Stad Gent"
  }
}
```

```
<P1>
  <name>
    Vrijdagmarkt
  </name>
  <parkingSpaces>
    648
  </parkingSpaces>
  <contact>
    Stad Gent
  </contact>
</P1>
```

```
id,name,parkingspaces,contact
P1,Vrijdagmarkt,648,Stad Gent
```

Figure 1. An example dataset respectively in JSON, XML and CSV.

As humans decide how datasets are shaped, human language is used to express facts within these serializations. Noam Chomsky, who laid the foundations of generative grammar, built models to study language as if it were a mathematical problem. In Chomskian generative grammar, the smallest building block to express a fact one can think of, is a triple, containing a *subject*, a *predicate* and an *object*, in which the subject has a certain relation to an object, and this relation is described by the predicate. In Figure 2, we illustrate how our CSV example earlier would look like in a triple structure.

```
P1 → name → Vrijdagmarkt
P1 → parkingSpaces → 648
P1 → contact → Stad Gent
```

Figure 2. Three triples which can be extracted from the example dataset in Figure 1.

This triple structure – rather than a tabular or hierarchical data model – helps studying data using its most basic building blocks. It is also the basis of graph-based data models, as each element in a triple can also be used in another triple, and is able to link together statements. Different dedicated serializations for triples exist, such as Turtle [6] and N-Triples [7]. Also existing serializations such as JSON or XML can be used to encode triples within a document [8].

Semantic interoperability

Imagine the three triples in Figure 2 are published by three different machines, by three different organizations. When a user agent, visits these three machines separately, it can now answer more questions than each of the machines would on their own. However a problem occurs: how does this user agent know how each term relates to another? Vrijdagmarkt could as well be a Park or a Point of Interest identifier.

Instead of using words to identify things, information architects propose to use unambiguous identifiers, such as a number to identify things. This way, every organization can have their context in which entities are described and discussed. E.g., in this example the Vrijdagmarkt Parking Site is given the identifier P1. Yet for an outsider, it becomes unclear what the meaning is of P1, as it is unknown where its semantics are documented, if documented at all.

Linked Data solves this problem by using Web identifiers, or HTTP URIs. It is a method to distribute and scale semantics over large organizations such as the Web. When looking up this identifier – by using the HTTP protocol or using a Web browser – a definition must be returned, including links towards potential other interesting resources. The triple format to be used in combination with URIs is standardized within the RDF. In Figure 3, we exemplified how these three triples would look like in RDF.

```
<https://gent.stad/id/parking/P1> <http://xmlns.com/foaf/0.1/name> “Vrijdagmarkt” .
<https://gent.stad/id/parking/P1> <http://linkedgeodata.org/ontology/capacity> 648 .
<https://gent.stad/id/parking/P1> <http://linkedgeodata.org/ontology/operator>
<http://sws.geonames.org/2797656/>.
```

Figure 3. Three triples made semantically interoperable using URIs.

The URIs needed for these triples, except for a URI for Parking Site P1, already existed in other data sources, and we thus favored using the same identifiers. It is up to a data publisher to make a choice on which data sources can provide the identifiers for a certain of entities. In this example, we found Geonames to be a good source to define the administration of Ghent, we found the Linked Geo Data ontology fit for describing basic parking site properties, and we found the friend of a friend (FOAF) vocabulary a good URI namespace for defining terms like “name”.

The city of Ghent

The mobility organization of Ghent was given the task to build a virtual traffic centre. The centre should inform Ghentians with the latest up to date information about mobility in the city. As the city is in a transition, banning cars from the city centre, this is a project with high expectations. Information to build this traffic centre comes from existing GIS layers, containing the on-street parking zones with, among others, their tariffs, all off-street parking lots, all streets and addresses or all traffic signs. Also real-time datasets are available, such as the sensor data from the induction loops, bicycle counters, thermal cameras and the availability of the off-site parking lots. Third parties also contribute datasets, such as the public transit time schedules and their real-time updates, or traffic volumes in, to and leaving the city. In order to bring these datasets from various sources together, and analyse them, both the semantics as the queryability is lacking.

Publishing data about parking facilities

In Ghent, a URI strategy is in place to negotiate identifiers since 2016. The URI strategy defines a base URI at “https://stad.gent/id/”. Using this strategy, the city introduced URIs for each parking site, similar to the examples above. When we now would point our browser to https://stad.gent/id/parking/P1, we will be directed to a page about this parking space. Furthermore, this identifier is interoperable across different systems, as when we would GET this URI from a computer program, I can negotiate its content type, and request an RDF representation of choice.

Next, in a test environment, we published a Linked Data document at http://linked.open.gent/parking/, by transforming the current real-time XML-feeds using a PHP-script (source code is made available at https://github.com/opentransportnet/ghent-datex2-to-linkeddata). First, this script adds metadata to this document indicating this data has an open license, and is thus legally compliant to the Open Definition (http://opendefinition.org). Next, we added HTTP headers, indicating that this document can be used for Cross Origin Resource Sharing (CORS), as well as an HTTP header that this document can be cached for 30 seconds. Lastly, we also added a content negotiation for different RDF representations.

As we would like to solve Basic Graph Pattern (BGP) queries, we added the hypermedia controls requested by the Triple Pattern Fragments (TPF) specification, which details how to filter the triples in this document based on their subject and/or predicate and/or object [9]. As the number of data facts that need to be published is small enough, we directed all controls to the main document, and did not expose extra server functionality. The latest version of the Triple Pattern Fragments hypermedia specification can be found at https://www.hydra-cg.com/spec/latest/triple-pattern-fragments/.

Following these steps, we made the data *queryable* through clients that can exploit the Triple Pattern hypermedia controls, such as the Linked Data Fragments client available at http://client.linkeddatafragments.org. This client is able to query over multiple Triple Pattern Fragments interfaces at once, and thus answer federated queries by following hypermedia controls. We present a demonstration query can now be answered on top of the real-time Linked Data. The following query selects the name, the real-time available and the geolocation from Open Street Map (Linked Geo Data) of a parking lot in Ghent with more than 200 spaces available: http://bit.ly/2jUNnES. This demonstrates that complex questions can still be answered over simple interfaces. The overall publishing approach is cost-efficient and stays as close as possible to the HTTP protocol as a *uniform interface*. The only part where an extra technical investment was needed was in documenting the definitions of the new URIs for the parking sites.

This query can then easily be used by Graphical User Interfaces, as exemplified in Figure 4.

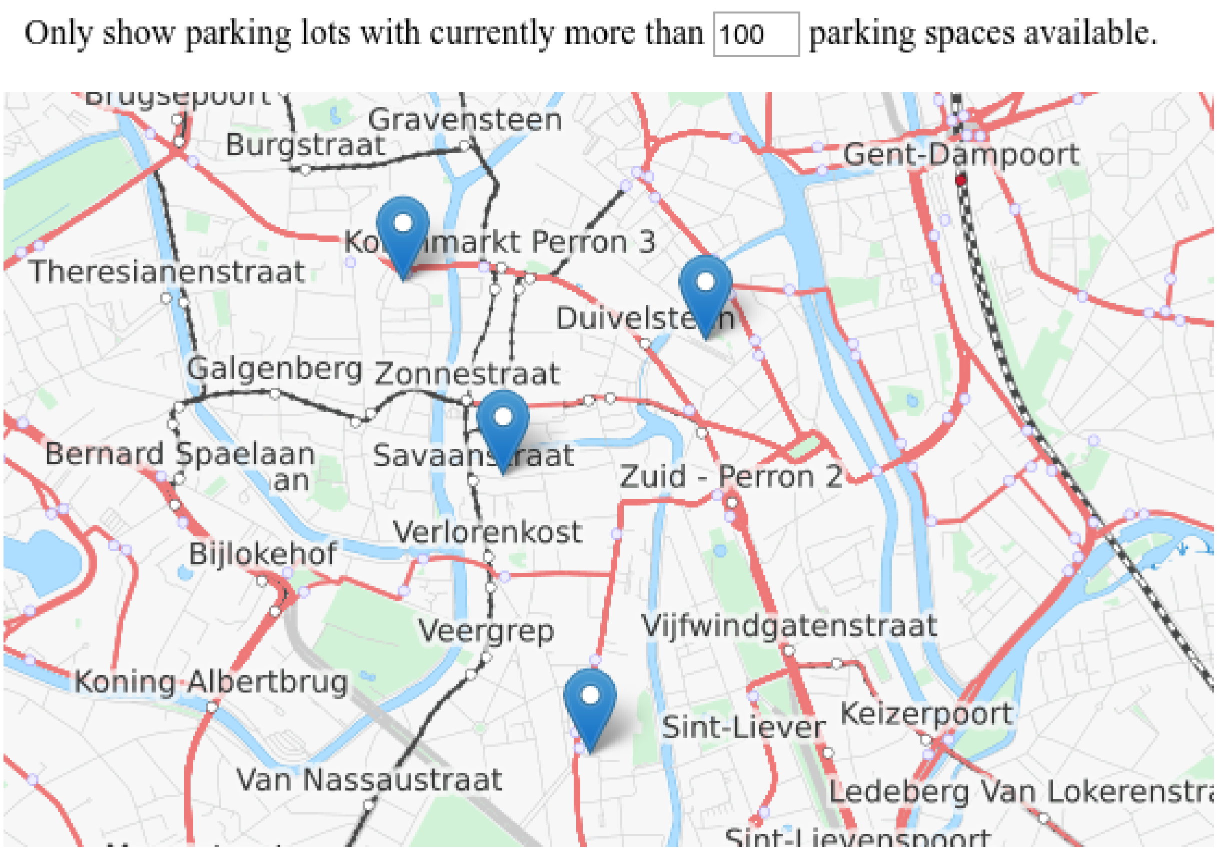
Only show parking lots with currently more than 100 parking spaces available.

Figure 4. A demo showing the location of parking lots from one dataset, while retrieving the real-time availability of another using the Linked Data Fragments client [8], available at http://codepen.io/pietercolpaert/pen/wgoPNV

DATEX2

DATEX2 (http://www.datex2.eu) refers to a set of European (CEN) standards that was originally developed to publish and exchange traffic information. Several extensions have in the meanwhile been added, such as the parking extension used in the example above. CEN standards are developed based on active participation of the key stakeholders in EU Member States. Though time-consuming, this consensus-building process pays off as the resulting standards are often imposed via EU legislation.

DATEX2 specifies the shape of XML files for data exchange in Europe. To that extent, it documents the terms it uses in these XML elements as part of an XML schema. The domain model would be a big contribution for Web of data. To that extent, we exemplified a mapping of DATEX2 to Linked DATEX2 at http://vocab.datex.org/terms.

Conclusion

The goal of an Open Data policy is to share data with anyone for any purpose. The intention is to maximize the reuse of a certain data source. When a data source needs to attract a wide variety of use cases, this data source need to be simple to adopt, needs to be reliable, and needs to be cost-efficient to host. Clients and servers implement the HTTP protocol so that their communication is technically interoperable. Using URIs for the identifiers, every element is documented using a *uniform interface*. Furthermore, all documents can also be reached through this interface, in which links, and other hypermedia controls, provide the engine of the application's state. To that extent, HTTP, Linked Data and hypermedia form the perfect building blocks for a scalable machine documented information system.

As a demonstrator, we applied the REST constraints to the use case of publishing a data source on parking availability in Ghent. While a SOA approach was closer to the way the city of Ghent was used to work, it did prove to be a task without big new investments. The demonstrator shows that applying the REST constraints comes with benefits, such as server scalability and a good user-perceived performance. Furthermore, the Linked Data documents now allows anyone to build federated queries that go beyond its silo, without having to invest in CPU power for third parties. The biggest challenge left is to keep advocating the use of URIs in and beyond their own organization.

Today, specifications are created on top of HTTP, which are, like in the case of DATEX2, primarily documented for humans. By extracting the semantics from DATEX2 and using HTTP as a uniform interface, we can enhance, the reuse potential, reliability of the information system, scalability and discoverability of our resources. For future transport related specification, we advise enabling data publishers to also follow the REST constraints by providing URIs for the terms used in the domain model.

Acknowledgment

This work is a collaboration between Pieter Colpaert with the Open Transport Net project (http://opentransportnet.eu), Peter Van der Perre with a Flemish innovation study for ITS in Belgium (http://its.be), and Thimo Thoeye with the OASIS (https://oasis.team) project.

References

1. R. Fielding: “Architectural Styles and the Design of Networkbased Software Architectures”, (2000)
2. R. Verborgh, S. van Hooland, A.S. Cope, S. Chan, E. Mannens, R. Van de Walle: “The Fallacy of the Multi-API Culture: Conceptual and Practical Benefits of Representational State Transfer (REST)”, Journal of Documentation (2015)
3. T. Berners-Lee: “Information Management: A Proposal”, (1989)
4. Article 5 of Commission delegated Regulation (EU) No 885/2013 in the framework of the ITS Directive 2010/40/EU, http://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX%3A32013R0885, accessed 2d of October 2016
5. P. Colpaert, M. Van Compernolle, N. Walravens: “Open Transport Data as an enabler for multimodal route planning”, (2016) ITS-EU Congress 2016
6. D. Beckett, T. Berners-Lee, E. Prud’Hommeaux, G. Carothers: “RDF 1.1 Turtle – Terse RDF Triple Language”, W3C (2014)
7. G. Carothers, A. Seaborne: “RDF 1.1 N-Triples – A line-based syntax for an RDF graph”, W3C (2014)
8. G. Schreiber, Y. Raimond: “RDF 1.1 Primer”, W3C (2014)
9. R. Verborgh, M. Vander Sande,O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, P. Colpaert: “Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web”, Journal of Web Semantics (2016)