Optimising a Semantic IoT Data Hub

Ilias Tachmazidis, Sotiris Batsakis, John Davies, Alistair Duke, Grigoris Antoniou and Sandra Stincic Clarke

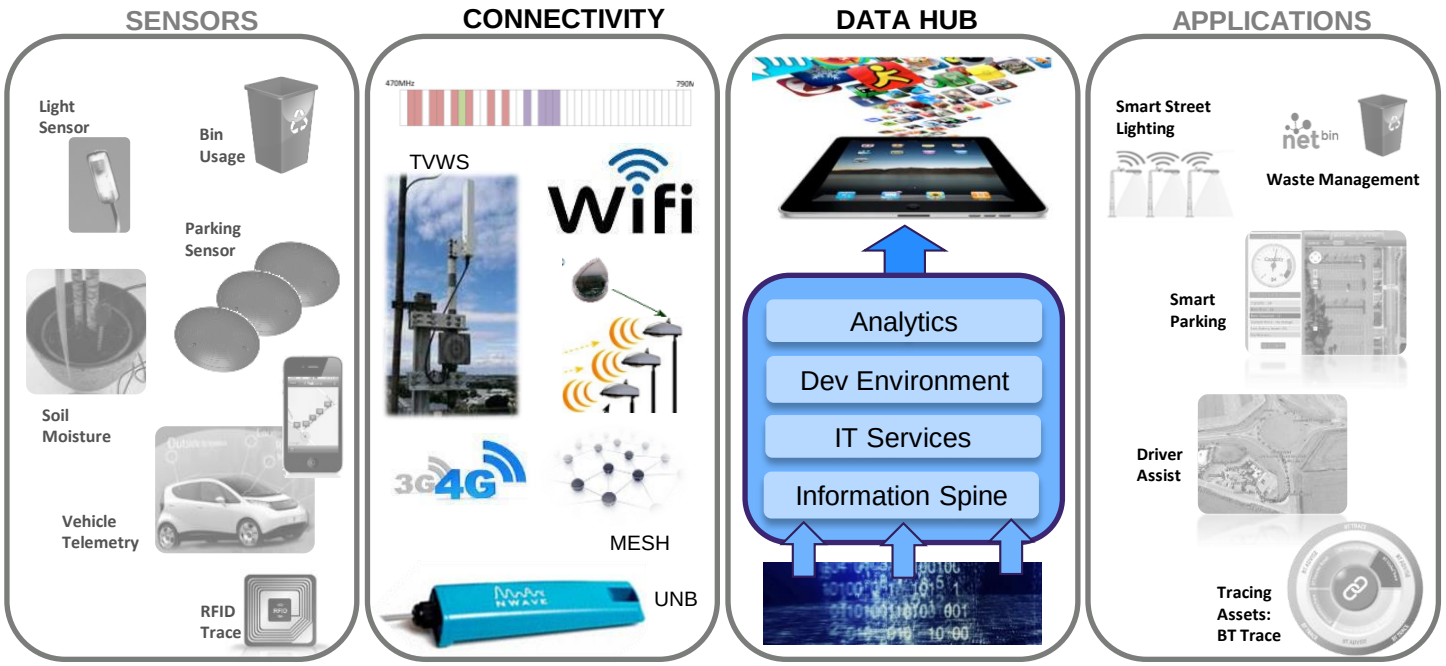
John Davies, BT

Overview

- Motivation – de-siloization and data hubs
- Semantic lifting of IoT Data hub
- Optimization of SPARQL – SQL mapping
- Concluding Remarks

BT

IoT Ecosystem



Enabling the IoT ecosystem

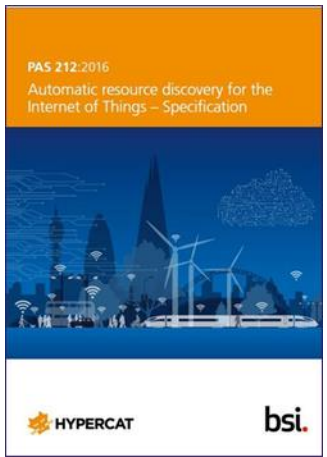
BT

Data interoperability & IoT hubs

Data hubs

- provide economies of scale and uniform access to data
- lower the barrier to participation, avoid silos and foster innovation
- ***Key challenges***
  - Resource discovery and access (what data does this hub have? how do I get it? what other hubs are there?)

Hypercat is a specification which attempts to address these issues



BT

HyperCat is:

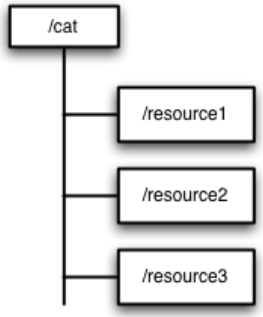
- A machine-readable catalogue format for IoT data
- A JSON file format for cataloguing IoT resources - *a HyperCat **file***
- A web API for fetching, serving, searching and updating HyperCat catalogues - *The HyperCat Web **API***

- A thin horizontal layer using technology well-understood by web developers
- Publically available specification from BSI, some open source tooling

BT

Catalogues and Resources

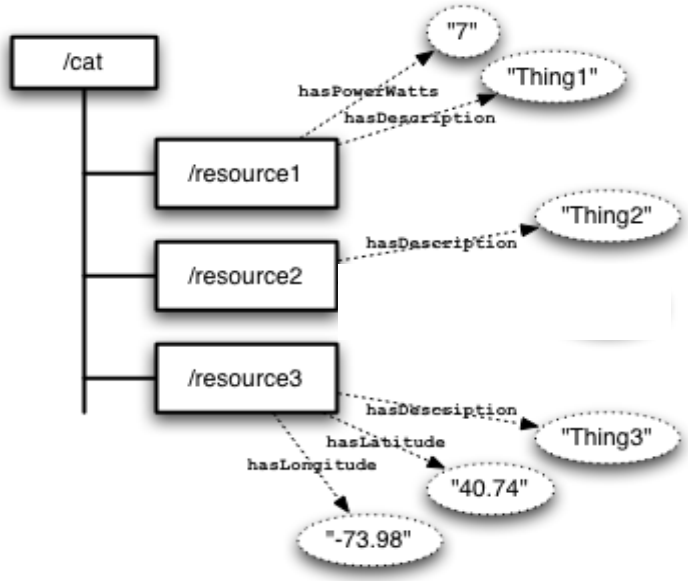
Servers (data hubs) provide catalogues of resources to clients
A catalogue is an array of URIs



BT

Resource Metadata

Each resource (URI) in the catalogue is annotated with metadata (triples!)



Hypercat fragment

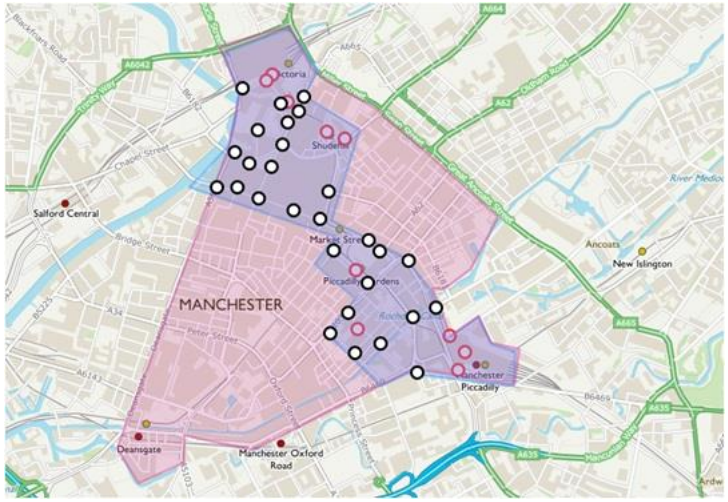
air quality data feed in Milton Keynes

```
"items":[
{
    "href":"http://api.stride-project.com/sensors/feeds/3bdae7b8-c4c6-
4701-81a7-e9ffcb47c6ac",
    "i-object-metadata":[
      {
          "rel":"urn:X-hypercat:rels:hasDescription:en",
          "val":"Air quality data from MK"
      },
      {
          "rel":"urn:X-hypercat:rels:isContentType",
          "val":"application/xml"
      },
```

Associating metadata with the feed id, a URI pointing to the feed data itself

Semantic lifting: why?
Case Study: CityVerve City Concierge

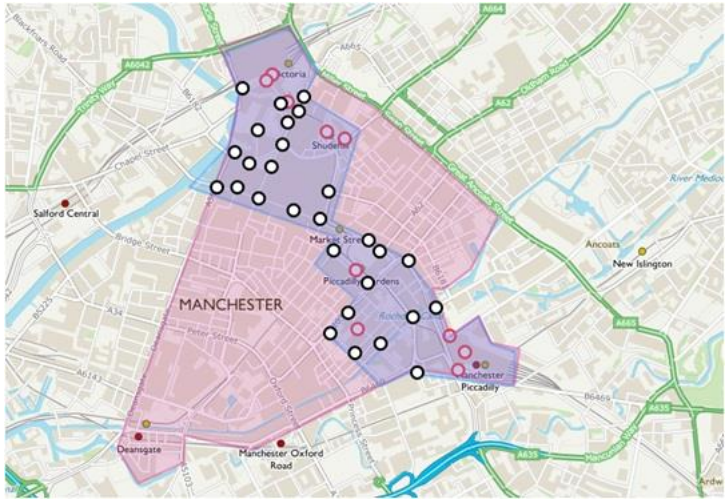
- Helping visitors to the city
  - Navigate & Access transport services
  - Find out about cultural events
  - Discover local businesses
  - Community feedback and asset management
  - Check the weather, air pollution, real-time traffic / travel



Semantic lifting: why?

Case Study: CityVerve City Concierge

- App delivery via smartphone, desktop and street furniture
- Requirement: combination of data from multiple sources
- (Federated) SPARQL queries allow integration of multiple data sources e.g. external endpoint about events can be queried

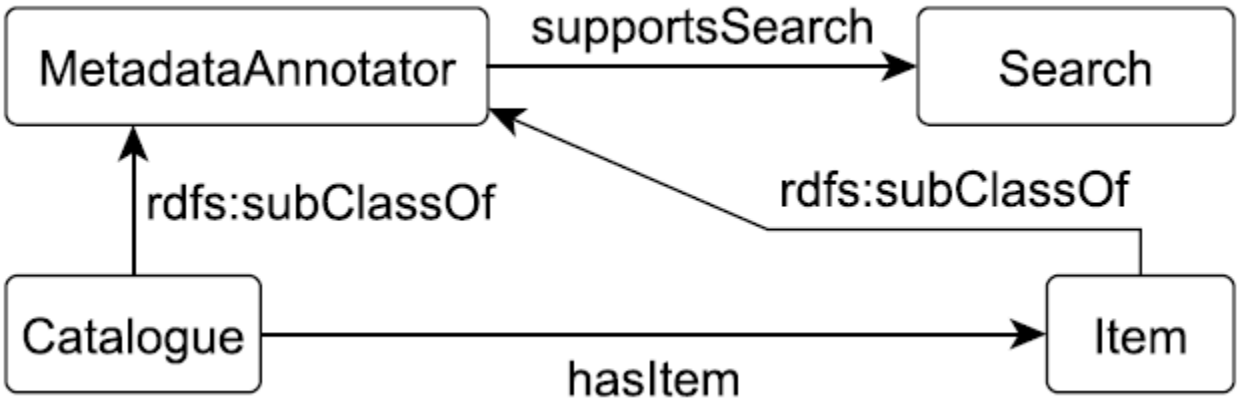


Semantic Enhancements to Data Hub : **Catalogue**
Semantic Lifting

- Existing DataHub provides:
  - Hypercat Catalogue using specified JSON format
  - Limited semantic capability
- Semantic enhancement allows:
  - Items in the catalogue to be related to existing (domain) ontologies supporting more powerful search and discovery
  - Reasoning over catalogue items
    - E.g. this feed about air temperature is also a feed about weather

BT

HyperCat Catalogue Ontology



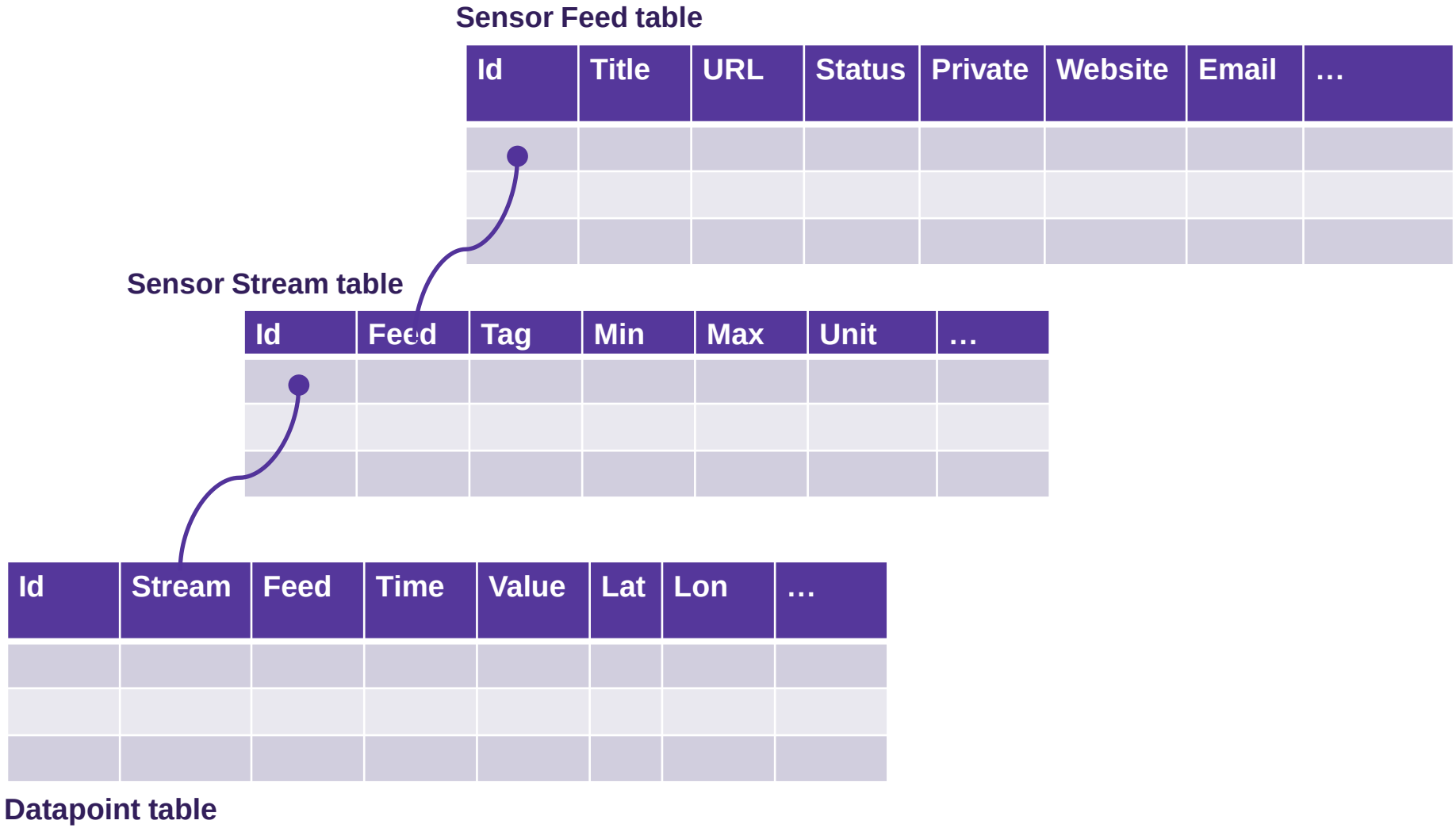
BT

Semantic Enhancements to Data Hub: **Data Feeds**
Semantic Lifting

- Existing DataHub provides:
  - Egress of data from SQL DB via RESTful API in XML or JSON format
- Semantic enhancement allows:
  - Data in the Data Hub to be related to existing (domain) ontologies supporting
    - Semantic enrichment (attaching ontological metadata, i.e. relate to other (external) classes or properties)
    - Rule-based reasoning (e.g. SWRL; define expected ranges, if exceeded, then trigger action)
    - Automatic translation of data to required form (e.g. units)
  - Querying across discrete data feeds to identify relevant data
  - Federated queries to additional external data (LoD Cloud)

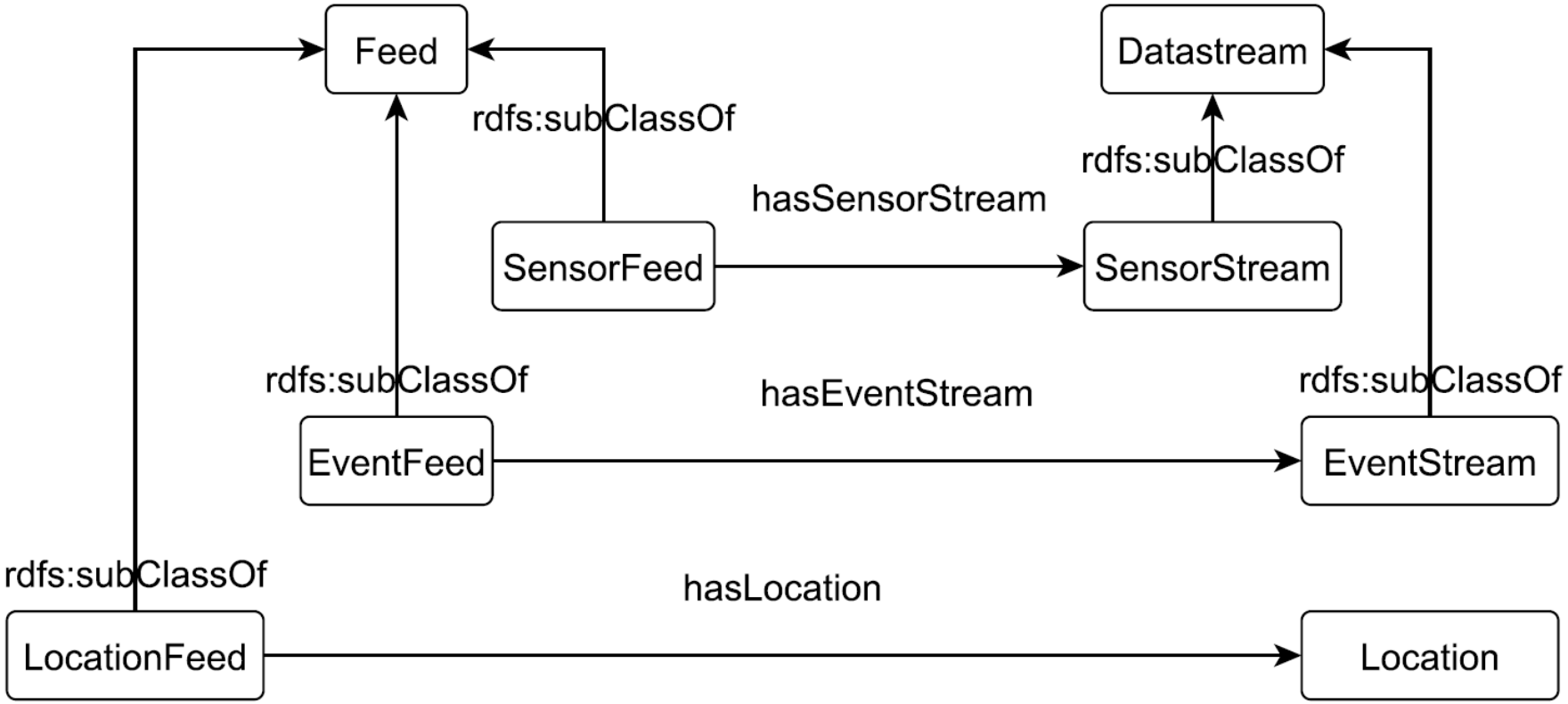
BT

Data hub tables (fragment) based on EEML (eeml.org)



BT

BT Data Hub Ontology (3 principal types: sensors, events, location data)



BT

HyperCat JSON to HyperCat RDF (N-Triples) (generate RDF version of catalogue)

http://portal.bt-hypercat.com/cat

“rel”: “urn:X-hypercat:rels:isContentType”

“val”: “application/vnd.HyperCat.catalogue+json”

<http://portal.bt-hypercat.com/cat-rdf>
<http://portal.bt-hypercat.com/hypercat#isContentType>
“application/n-triples" .

BT

Data Translation

- RDF triple is provided within a single line, in N-Triples format, namely:
  - <subject> <predicate> <object>.
- The URI of a **SensorFeed** is generated by BT data hub as:
  - http://api.bt-hypercat.com/sensors/feeds/feedID

- An RDF triple providing the type of a **SensorFeed**:

<http://api.bt-hypercat.com/sensors/feeds/feedID7>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://portal.bt-hypercat.com/ontologies/bt-hypercat#SensorFeed>

BT

HyperCat properties and their ontological equivalents (rels -> properties)

JSON-based	RDF-based
urn:X-hypercat:rels:hasDescription:en	rdfs:comment
urn:X-hypercat:rels:supportsSearch	hypercat:supportsSearch
urn:X-hypercat:rels:isContentType	hypercat:isContentType
urn:X-hypercat:rels:hasHomepage	hypercat:hasHomepage
urn:X-hypercat:rels:containsContentType	hypercat:containsContentType
urn:X-hypercat:rels:hasLicense	hypercat:hasLicense
urn:X-hypercat:rels:acquireCredential	hypercat:acquireCredential

MetadataAnnotator Properties

BT

SPARQL to SQL translation

- Uses Ontop Library
  - http://ontop.inf.unibz.it/

- Protégé plugin used to map SPARQL patterns to SQL queries
  - Includes reasoner
    - Parses mappings and ontology
    - Translates queries to SQL

- Apache Jena used to offer federated queries

BT

Semantic Querying - SPARQL to SQL

- Dynamic translation of SPARQL queries into SQL, using Ontop
  - OBDA tool - http://ontop.inf.unibz.it/
- Implicit information is extracted from the ontology through reasoning
- Where richer domain ontologies are linked, semantically richer information is extracted compared to the knowledge that is stored in the relational database

BT

Data Translation - SPARQL to SQL

- Mapping ID: a unique id for a given mapping,
- Target (Triple Template): RDF triple pattern to be generated in the answer (SQL variables are given in braces, such as {feed.id})
- Source (SQL Query): SQL query to be created and submitted to the relational database

Mapping ID	mapping:SensorFeed
Target (Triple Template)	bt-sensors:feeds/{feed.id} a bt-hypercat:SensorFeed
Source (SQL Query)	SELECT feed.id FROM feed

URI prefixes:

bt-sensors: http://api.bt-hypercat.com/sensors/

bt-hypercat: http://portal.bt-hypercat.com/ontologies/bt-hypercat#

BT

Mapping Example

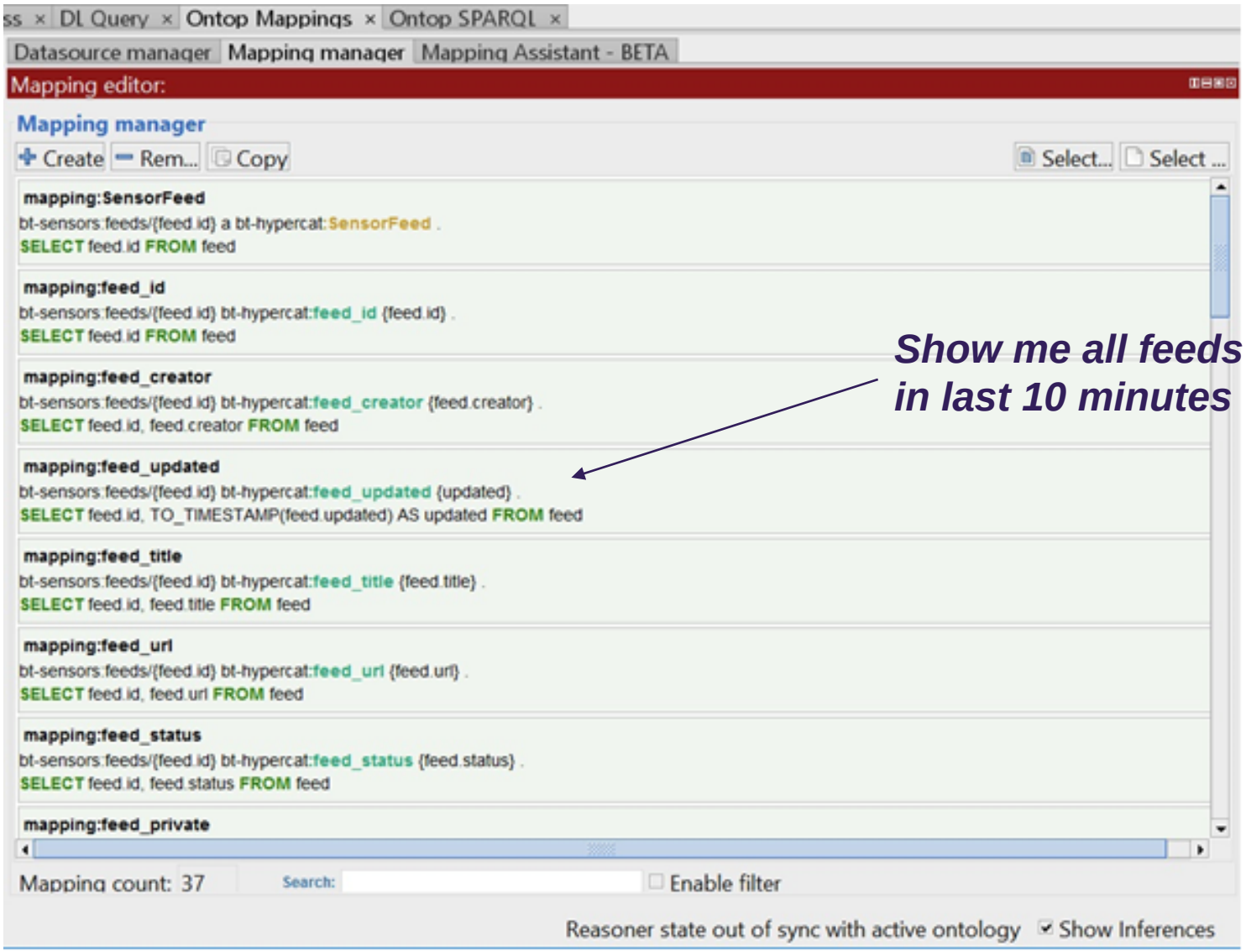
Mapping ID	Mapping:SensorFeed
Target (triple template)	bt-sensors:feeds/\{feed.id\} a bt-hypercat:SensorFeed .
Source (SQL Query)	SELECT feed.id FROM feed

```
PREFIX hypercat: <http://portal.bt-hypercat.com/ontologies/bt-hypercat#>
SELECT DISTINCT ?s
WHERE{ ?s a hypercat:Feed . }
```

- Ontop will match the triple pattern ``?s a hypercat:Feed'' with the mapping ``mapping:SensorFeed'' since class SensorFeed is subclass of Feed (and would also map to equivalent mapping:EventFeed)
- An SQL query will be submitted to the relational database, while the retrieved Ids (feed.id ?s) will be used in order to generate the response as RDF triples following the triple template
- SQL not aware of subclasses of feed, so sensorfeed is mapped to feed

BT

Ontop Protégé plug-in



Show me all feeds updated in last 10 minutes

Complex Query Example

Query description: *Get distinct resources of type **"Datapoint"**, along with data properties **"datapoint_at_time",** and **"datapoint_value"** that are related to them, where **"datapoint_at_time"** is dated between "2018-03-23T09:00:00Z" and "2018-03-23T10:00:00Z"*

Query text:

```
PREFIX hypercat: <http://portal.bt-hypercat.com/ontologies/bt-hypercat#>
SELECT DISTINCT ?s ?o1 ?o2
WHERE{ ?s a hypercat:Datapoint .
?s hypercat:datapoint_at_time ?o1.
?s hypercat:datapoint_value ?o2.
FILTER (?o1>"2018-03-23T09:00:00Z"&&?o1 <"2018-03-23T10:00:00Z")
}
```

Mapping ID	mapping:datapoint_value
Target (Triple Template)	bt-sensors:feeds/{datapoint.feed}/datastreams/{datapoint.stream}/datapoints/{at_time} bt-hypercat:datapoint_value {datapoint.val} .
Source (SQL Query)	SELECT datapoint.feed, datapoint.stream, TO_TIMESTAMP(datapoint.at_time) AS at_time, datapoint.val FROM datapoint

Architectural Options

- **Replication** of data in triple store

  + Queries require no translation so should be faster

  - IoT sensor data quickly becomes out of date

  - Additional storage and maintenance burden

- **On demand** query / result translation

  + No additional storage required

  + Freshest data always available

  - ***Translation overhead means slower responses to queries***

BT

Optimising SPARQL to SQL endpoint

BT

Ontology Optimizations

- Class hierarchy restricted to classes explicitly used by Ontop for mappings
  - faster reasoning
  - subclasses of Feed such as PublicTransportFeed also of Stream – not used in mappings and were removed
- Similarly, property hierarchy restricted to properties explicitly used by Ontop for mappings
- Domain and range assertions deleted from properties where classes are already mapped, ensuring:
  - duplicate reduction
  - eg Datafeed class and a hasDatastream property, do not define domain to be datafeed since duplicates will be generated
  - Ontop attempts to extract as much information as possible during the translation from SPARQL to SQL. Any possible mapping that could derive Datafeed, will be translated into yet another SQL query

BT

SQL Plans Optimizations

- In Ontop, mappings with SQL functions such as:
  - *TO_TIMESTAMP()* for time
  - *unnest()* for arrays
  - *ST_AsText()* for PostGIS geometry

  are translated into separate subqueries

- Such subqueries may be inefficient, since:
  - they are not indexed (executed over temporary SQL tables)
  - they can lead to unnecessary self-joins

BT

SQL Plans Optimizations

- Solution:
  - DB columns representing **time** or **PostGIS geometry** are translated into a **simpler form** (such as WKT string representation for geom objects)
  - DB columns representing **arrays** need to be stored in a **separate SQL table**
  - Translate mappings that require the use of SQL functions into mappings over SQL tables with simple data forms

BT

SQL Table Transformation

- Initial SQL table:
  - TABLE **feed**(id uuid NOT NULL, updated **bigint**, tag **character varying[]**, the_geom **geometry**)
- Translated to 2 new tables
  - TABLE **sparql_feed**(id uuid NOT NULL, updated **character varying**, the_geom **character varying**);
  - TABLE **sparql_feed_tag**(id uuid NOT NULL, tag **character varying** NOT NULL); (one of these per array entry)
- More efficient because array processing etc is eliminated
- Processing cost moves from query time to assertion time

BT

SQL Data Transformation (1/2) to generate new table

- INSERT INTO sparql_feed (id, updated, the_geom)
  SELECT id,
    TO_TIMESTAMP(feed.updated) AS updated,
    ST_AsText(feed.the_geom) AS the_geom FROM feed

BT

SQL Data Transformation (2/2) to generate new table

- INSERT INTO sparql_feed_tag (id, tag)
  SELECT feed_tag.id, feed_tag.tag
  FROM (SELECT feed.id,
  unnest(feed.tag) AS tag
  FROM feed) AS feed_tag

BT

Mappings Optimization (1/2)

- Target (Triple pattern):

  bt-sensors:feeds/{**sparql_feed.id**}

  bt-hypercat:feed updated

  {sparql_feed.updated} .

- *Sparql_feed.id* is the **key** of SQL table *sparql_feed*

- *Query is looking for feeds, therefore feed.id column should be the table key*

BT

Mappings Optimization (2/2)

- If the key contains more columns than those used in the RDF triple pattern to be generated, then:
  - self-joins cannot be eliminated
  - each mapping is translated as a separate subquery
- Self-joins can be:
  - **Manageable** for relatively **small tables** (containing thousands of rows, if the table is **indexed**)
  - **Prohibitive** for **large tables** (containing millions of rows)

- *If you have a key across 2 columns (eg) but only 1 column used to generate URI (i.e. feed.id), self-joins required*

BT

SPARQL Query Optimizations (1/2) – expensive operations

- **Where possible, specify the predicate** within each triple pattern, since triple patterns such as "?s ?p ?o":
  - will be translated using all available mappings (a UNION of all defined mappings)
  - will lead to an excessive SQL query plan

- **Where feasible, avoid DISTINCT** as:
  - performance deteriorates
  - final results are sorted and filtered for unique values at the end of the SQL query plan

BT

SPARQL Query Optimizations (2/2)

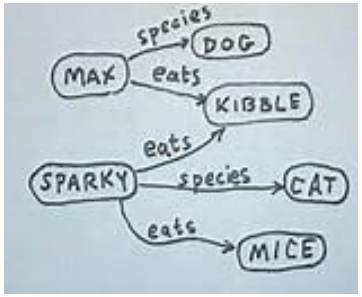
- **Use FILTER** to retrieve specific Feeds:
  - Restricting the search space early leads to more efficient SQL query plans.

- **Avoid OPTIONAL** as:
  - each OPTIONAL is translated into a LEFT OUTER JOIN
  - If used, OPTIONAL should be put at the end of the query

- **Use LIMIT** as limiting the amount of required results could speed up the query execution

BT

Semantic Technology and IoT



- W3C, IEEE, oneM2M and AIOTI
- Joint White Paper ***Semantic Interoperability for the Internet of Things***
  - *"The **full** value potential can only be unlocked if … ad-hoc, cross-domain systems of IoT are able to establish conversations and build understanding"*
- Using semantic technology to provide machine-processable, shared vocabularies with automated reasoning
- Hypercat – lightweight, domain-independent
- Hypercat-RDF – richer descriptions, link to domain-specific vocabularies, federated querying, reasoning

BT

Concluding Remarks

- Data hubs can help remove silos and maximise value of data
  - There may be additional value in using semantic tech
- SPARQL to SQL can be *very* slow
- Significant speed-up is possible
  - Reliant on the ability/willingness to manually inspect and edit the ontology and underlying database
- Next steps
  - quantify the efficiency improvements
  - identify further use cases where semantic tech adds value
  - (understand relationship to SSN work!!)

BT

Bringing it all together

Thanks for your attention

john.nj.davies@bt.com

BT