# Objects as results from graph queries using an ORM and generated semantic-relational binding

Marc-Antoine Parent
maparent@acm.org

Imagination for People

**Abstract.** This poster describes a method to expose data in a object-relational model as linked data. It uses Virtuoso's Linked Data views on relational data to expose and query relational data. As our object-relational model is evolving, we generate the linked data view definition from augmented ORM declarations.

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement n°6611188.

## 1 Interoperability in the Catalyst Consortium

The Catalyst consortium[1] has been funded by the European Commission to develop existing tools for argument mapping and argument analytics into an ecosystem of collective intelligence tools [2] using Linked Data, and test those tools on a large scale.

The existing tools (Cohere[2] by Open University's KMI[3], Deliberatorium[4] by MIT's Centre for Collective Intelligence[5] and University of Zürich[6], EdgeSense[7] by Wikitalia[8], and Assembl[9] by Imagination for People[10]) have used a disparate array of languages (PHP, Lisp, Python) and relational databases (PostgreSQL, MySQL). More important, the data models differ in many ways: Idea-centric or message-centric, importance of data history, of social analytics, idea classification at idea creation or *post-hoc*, etc. Finally, we were all dealing with algorithmic problems that we knew could benefit from graph queries in semantic databases.

[1] http://catalyst-fp7.eu/
[2] http://cohere.open.ac.uk/
[3] http://kmi.open.ac.uk/
[4] http://cci.mit.edu/klein/deliberatorium.html
[5] http://cci.mit.edu/
[6] http://www.ifi.uzh.ch/index.html
[7] https://github.com/Wikitalia/edgesense
[8] http://www.wikitalia.it/
[9] http://assembl.org/
[10] http://imaginationforpeople.org/fr/

The technical partners agreed to use Linked Data technologies for interoperability, both for its inherent flexibility and because there were relevant standard ontologies that could be leveraged. We co-designed a format [6] that could accommodate the model diversity, leveraging a few common ontologies, such as SIOC, OpenAnnotation, and FOAF; and with equivalences to other relevant ontologies such as AIF. New ontologies were developed when necessary, for example for IBIS data. To lower the barrier to entry for partners with limited expertise with Semantic Web technologies, we agreed on RESTful exchange of JSON-LD data as the main interchange protocol. Partners with legacy relational models could enter the ecosystem with simple JSON parsers and generators.

The Assembl platform could not follow that simple model: our legacy model was object-oriented, using SQLAlchemy [1], a Python declarative ORM, with PostgreSQL. We wanted to use a semantic database rather than a semantic wrapper on a traditional relational database, so we could display the results of complex graph queries efficiently. On the other hand, we had a fair amount of business logic coded at the object layer, which we wanted to leverage; and the object model was under continuous development, and we did not want to maintain a semantic-relational wrapper independently.

## 2 Existing solutions for proxies to data storage

The first alternatives we rejected were: Pure relational (weakness of graph-oriented queries), pure semantic (relative obscurity of object-semantic tooling in Python), and partitioning our data between two databases, with the relationships in a semantic database and the content in a RDBMS (overhead of joining across database systems).

When dealing with relational data, Object-Relational Mappings (ORMs) allow developers to write an object model annotated with mappings to the relational model. This annotated object model can act as a OO wrapper, or proxy to the relational model. Many ORMs also allow developers to generate the relational model from the object model, or more rarely the object model (with relational annotations) from the relational model through relational introspection. In either case, we have a single authoritative model, which software engineers also call the "don't repeat yourself" (DRY) principle.

With semantic data, we also have object proxies over semantic data[11]. As with an ORM, OO code can be annotated with semantic mapping annotations, as with RDFAlchemy[12] or Elmo[13]) Also similarly, the OO code of those proxys can be generated from the RDFS or OWL models, as with RdfReactor[14] or Jastor[15] respectively, and others [5]. In the case of dynamic languages like Python, it is

[11] http://semanticweb.org/wiki/Tripresso
[12] http://www.openvest.com/trac/wiki/RDFAlchemy
[13] http://www.openrdf.org/elmo.jsp
[14] http://rdfreactor.semweb4j.org/
[15] http://jastor.sourceforge.net/

also possible to dynamically translate accessor queries to unchecked RDF access, as with, for example, SuRF[16] or OldMan[17].

Another bridging technique involves semantic mapping over relational data. An adapter will use this mapping to act as a semantic proxy. The most well-known mapping language is the R2RML standard [3], but Virtuoso offers its own Linked Data Views syntax [4]. Those technologies allow great flexibility, but require to maintain the semantic-relational mapping in synchrony with the pre-existing semantic and relational models.

## 3 Generated semantic-relational binding for Assembl

We opted to use the Virtuoso database and enrich the relational annotation layer of SQLAlchemy with a semantic layer, with enough information to generate not only the relational model, but also the semantic mapping. This gives us both OO and semantic proxies over relational data. Simple traversals are converted by the ORM into relational queries, while more complex traversals are written as embedded SPARQL. We can expose the data as a SPARQL endpoint, or export it as JSON-LD for the benefit of the Catalyst ecosystem.

We generate Virtuoso linked data view definitions rather than R2RML mappings (which our annotations would also allow.) This allows us to also exploit Virtuoso's capability to embed a SPARQL subquery in a SQL query. Thus, our code receives ORM objects directly from a SPARQL query, and a single code-base serves as both an OO and semantic proxy to our data. We still have to keep this annotation layer up-to-date with both our relational and semantic models, as in the case of a hand-crafted semantic-relational mapping; but we avoid the maintainability cost of updating a distinct OO layer.

The poster[18] contains example of data definitions in the object model, and their translation to a Virtuoso linked data binding.

*Implementation* Our semantic annotation layer[19] is based on work by William Waites[20] to extend SQLAlchemy with specificities of the Virtuoso SQL dialect. An extensible introspector visits the classes known to the ORM and obtains the following information from class and columns annotations: the IRI pattern for the class; a global condition for the class to be included in the mapping; and for each database column, a specification needed to define a specific quad in the Linked data view.

SQLAlchemy has advanced capability to translate Object-Oriented (OO) inheritance into complex relational patterns[21], so the introspector has to cater to class definitions spanning many tables, or multiple classes sharing a single table,

[16] https://pythonhosted.org/SuRF/
[17] https://github.com/oldm/OldMan
[18] http://maparent.ca/iswc2014poster.pdf
[19] https://github.com/maparent/virtuoso-python
[20] http://river.styx.org/ww/2010/10/pyodbc-spasql/index
[21] http://docs.sqlalchemy.org/en/rel_0_9/orm/inheritance.html

and generate appropriate bindings. In Assembl, a subclass of the introspector also allows more quad specifications tied to more than one column, multiple graphs, global conditions that apply to class patterns, etc.

Much of the quad specification besides the predicate can be left blank, as the introspector can be initialized with a default graph, the subject is the class' subject IRI pattern, and the object is the column to which the quad specification is attached, which is interpreted to be either a literal or the application of an IRI pattern which can be inferred from foreign key information.

The quad specification may also specify a condition of applicability using the ORM constructs. The condition's structure is visited to define a coherent set of table aliases for this condition, which will be used in the linked data binding. A reference to a column defined in a superclass (which may appear in the object or condition of the quad specification) will enrich the condition with the appropriate table join; similarly, a reference to a subclass which does not define its own table will re-use the appropriate ORM condition.

## 4 Open issues

Having exposed our relational data as linked data, we will next work on importation of semantic data, and translating it into our relational model. We have to contend with the fact that open-world semantic data may not conform to referential integrity constraints defined at the relational layer. Also, because it is based on SQLAlchemy models, our solution follows its OO model with single inheritance.

## References

1. Bayer, M.: Sqlalchemy. In: Brown, A., Wilson, G. (eds.) The Architecture of Open Source Applications: Elegance, Evolution, and a Few More Fearless Hacks., vol. 2. Lulu.com (2012), `http://aosabook.org/en/sqlalchemy.html` 2
2. Buckingham Shum, S., De Liddo, A., Klein, M.: Dcla meet cida: Collective intelligence deliberation analytics. The Second International Workshop on Discourse-Centric Learning Analytics (Mar 2014), `http://dcla14.files.wordpress.com/2014/03/dcla14_buckinghamshumdeliddoklein1.pdf` 1
3. Das, S., Sundara, S., Cyganiak, R.: R2rml: Rdb to rdf mapping language. W3c recommendation, World Wide Web Consortium (September 2012), `http://www.w3.org/TR/2012/REC-r2rml-20120927/` 3
4. Haynes, T.: Mapping relational data to rdf with virtuoso's rdf views. Tech. rep., OpenLink Software (2010), `http://virtuoso.openlinksw.com/whitepapers/Mapping%20Relational%20Data%20to%20RDF.pdf` 3
5. Kalyanpur, A., Pastor, D.J., Battle, S., Padget, J.A.: Automatic mapping of owl ontologies into java. In: Maurer, F., Ruhe, G. (eds.) SEKE. pp. 98–103 (2004) 2
6. Parent, M.A., Grégoire, B.: Software architecture and cross-platform interoperability specification. D 3.1, Catalyst-FP7 (Mar 2014), `http://catalyst-fp7.eu/wp-content/uploads/2014/03/D3.1-Software-Architecture-and-Cross-Platform-Interoperability-Specification.pdf` 2