GIZE: A Time Warp in the Web of Data

Valeria Fionda[1], Melisachew Wudage Chekol[2], and Giuseppe Pirrò[3]

[1] University of Calabria, Italy
fionda@mat.unical.it
[2] University of Mannheim, Germany
mel@informatik.uni-mannheim.de
[3] Institute for High Performance Computing and Networking, ICAR-CNR, Italy
pirro@icar.cnr.it

Abstract. We introduce the GIZE framework for querying historical RDF data. GIZE builds upon two main pillars: a lightweight approach to keep historical data, and an extension of SPARQL called SPARQ–LTL, which incorporates temporal logic primitives to enable a rich class of queries. One striking point of GIZE is that its features can be readily made available in existing query processors.

1 Introduction

Querying historical data is of utmost importance in many contexts, from city monitoring, where one needs to track different aspects (e.g., pollution, population) to generic exploratory research, where one is interested in posing queries like *"Retrieve players that are now managing some club they played for"* or *"Retrieve the annotation of a gene since the discovery of a particular interaction"*. The classical approach for querying RDF data (e.g., via SPARQL endpoints) only considers the latest version. In fact, querying of historical RDF data poses some challenges. The first concerns the representation and storing of historical data. Some approaches (e.g., [4, 7]) allow to retrieve data by providing timestamps. Other resort to dedicated indexing structures (e.g., [5]) to speed-up query processing. The second problem concerns what type of querying primitive to provide. The most common approach is to devise SPARQL extensions that work with intervals or SPARQL translations into temporal logic. The addition of ad-hoc components either in terms of data representation, query language or both, hinders the applicability on existing RDF (query) processing infrastructures.

To cope with these issues we present the GIZE[4] framework. GIZE is built around two main components. The first is a lightweight approach to store RDF data, where each version of the data is stored in a separate *named graph*. The second component is a powerful extension of SPARQL called SPARQ–LTL. This language inherits a variety of temporal operators from Linear Temporal Logics (LTL) [2]. To evaluate SPARQ–LTL on existing SPARQL processors we devise a translation to standard SPARQL queries.

[4] Gize (ጊዜ) is the Ethiopian word for time.

Related Work. Approaches like the DBpedia Wayback Machine [4] allow to retrieve data at a certain timestamp (provided by the user). Other approaches (e.g.,[7]) access historical data via (HTTP) content negotiation, typically using the Memento framework. Yet other approaches (e.g., [5]) introduce timestamps into RDF triples along with ad-hoc indexing strategies. In Description Logics, some proposals focus on temporal conjunctive query answering (e.g., [1]); other efforts (e.g., [6]) have focused on the translation of SPARQL into LTL. Surprisingly, the design of solutions for querying historical RDF data on *existing SPARQL processors* is still in its infancy. GIZE fills this gap by contributing an extension of SPARQL, called SPARQ–LTL, that allows for a rich class of temporal primitives borrowed from LTL (e.g., `SINCE`, `NEXT`, `PREVIOUS`) along with a translation from SPARQ–LTL queries into standard SPARQL queries.

2 The GIZE Framework

Representing Historical RDF Data. The tenet of GIZE is to enable querying of historical RDF data on existing processors. To represent versions, GIZE leverages the notion of RDF quad. An RDF quad (for simplicity, we omit bnodes) is a tuple of the form $\langle s, p, o, c\rangle \in \mathbf{I} \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{L}) \times \mathbf{I}$, where $\mathbf{I}$ (IRIs) and $\mathbf{L}$ (literals) are countably infinite sets. The forth element of the quad represents that *named graph* to which the triple belongs.

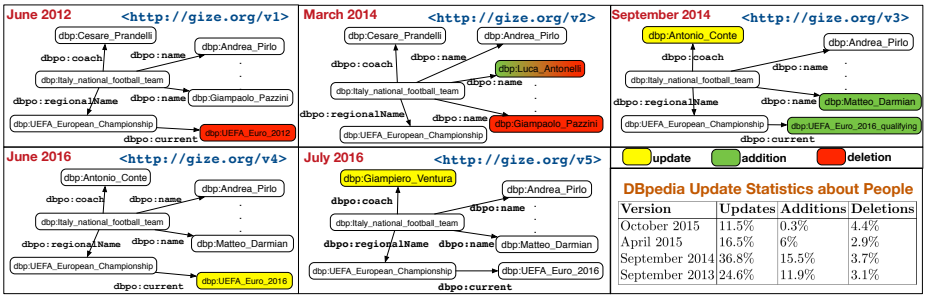

Fig. 1. An exceprt of evolving data from DBpedia.

Fig. 1 shows the evolution of some data taken from DBpedia. Each of the 5 versions considered is represented by a named graph. From this small data sample one may notice that Italy has changed 3 coaches from July 2012 (C. Prandelli) to July 2016 (G. Ventura). Interestingly, the latest coach (G. Ventura) will start on July 18th. One may also notice that some players like A. Pirlo were part of the team along the whole period, while some other like G. Pazzini or M. Darmiani were left out or added, respectively. The figure also shows (bottom right corner) update statistics about People in DBpedia. Each percentage is computed wrt the previous version. The availability of historical data allows to pose queries like *"Find all players that played with the highest number of coaches"* or *"Find players that played since C. Prandelli was the coach"*. Most of existing approaches either are not able to express such kind of queries or have to resort to ad-hoc processing infrastructures.

The SPARQ–LTL Language. The syntax of SPARQ–LTL is shown below while Table 1 provides a description of the temporal operators. Let q be a SPARQ–LTL query, $H = \{h_1, h_2, \ldots, h_m\}$ be the set of versions, and h_c be the current (not necessarily the latest) version of the data.

$$\begin{aligned}\mathsf{QP} ::= {} & \mathsf{QP}_1 \ .\ \mathsf{QP}_2 \mid \{\mathsf{QP}_1\}\ \textsf{UNION}\ \{\mathsf{QP}_2\} \mid \{\mathsf{QP}_1\}\ \textsf{MINUS}\ \{\mathsf{QP}_2\} \mid \mathsf{QP}_1\ \textsf{OPTIONAL}\ \{\mathsf{QP}_2\} \mid \\ & \mid\ \textsf{GRAPH}\ \mathbf{I} \cup \mathcal{V} \mathsf{QP}_1 \mid \mathsf{QP}_1\ \textsf{FILTER}\ \{R\} \mid t = (\mathbf{I} \cup \mathcal{V}) \times (\mathbf{I} \cup \mathcal{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathcal{V}) \mid \\ & \mid \mathsf{X}\{\mathsf{QP}\} \mid \mathsf{W}\{\mathsf{QP}\} \mid \mathsf{F}\{\mathsf{QP}\} \mid \mathsf{G}\{\mathsf{QP}\} \mid \{\mathsf{QP}_1\}\ \mathsf{U}\ \{\mathsf{QP}_2\} \mid \\ & \mid \mathsf{Y}\{\mathsf{QP}\} \mid \mathsf{Z}\{\mathsf{QP}\} \mid \mathsf{P}\{\mathsf{QP}\} \mid \mathcal{H}\{\mathsf{QP}\} \mid \{\mathsf{QP}_1\}\ \mathsf{S}\ \{\mathsf{QP}_2\}\end{aligned}$$

Operator	SPARQ–LTL **Syntax**	**Meaning**
X q	NEXT	Evaluate q on version h_{c+1}
F q	EVENTUALLY	Evaluate q on all versions $h_c, ...h_m$
G q	ALWAYS	The evaluation of q must be the same on all versions $h_c, ...h_m$
q_1 U q_2	UNTIL	If S_2 is the solution of q_2 in a version, $h_k \in \{h_c, ...h_m\}$ then there exists a solution S_1 of q_1 on h_c such that S_1 is compatible with S_2 in all versions $\{h_c, ...h_k\}$
Y q	PREVIOUS	Evaluate q on version h_{c-1}
P q	PAST	Evaluate q on all versions $h_1, ...h_c$
H q	ALWAYSPAST	The evaluation of q must be the same on all versions $h_1, ...h_c$
q_1 S q_2	SINCE	If S_2 is the solution of q_2 in a version, $h_k \in \{h_1, ...h_c\}$ then there exists a solution S_1 of q_1 on $\{h_k, ...h_c\}$ such that S_1 is compatible with S_2 in all versions $\{h_k, ...h_c\}$

Table 1. Meaning of the temporal operators in SPARQ–LTL.

SPARQ–LTL allows to use an additional set of keywords when writing SPARQL queries. SPARQ–LTL are evaluated by translating the temporal operators via a (set of) pattern(s) evaluated on named graphs maintaining data versions. We give some examples by considering the five version of data shown in Fig. 1 (stored in separate named graphs). In what follows, `dbp:INFT` is a shorthand for `dbp:Italy_national_football_team`.

Example 1. Select players who are playing in the Italian national football team and played at least under a different coach than the current one.

SPARQ–LTL

```
SELECT ?p WHERE {
  dbp:INFT dbpo:name ?p.
  dbp:INFT dbpo:coach ?c1.
  PAST{
    dbp:INFT dbpo:name ?p.
    dbp:INFT dbpo:coach ?c2.
    FILTER (?c1!=?c2)
  } }
```

Translation into SPARQL

```
SELECT ?p WHERE {
   dbp:INFT dbpo:name ?p.
       dbp:INFT dbpo:coach ?c1.
   {GRAPH <http://gize.org/v5> { dbp:INFT dbpo:name ?p.
   dbp:INFT dbpo:coach ?c2.
       FILTER (?c1!=?c2) } } UNION
{GRAPH <http://gize.org/v4> { dbp:INFT dbpo:name ?p.
    dbp:INFT dbpo:coach ?c2.
        FILTER (?c1!=?c2) } } UNION
{GRAPH <http://gize.org/v3> { dbp:INFT dbpo:name ?p.
    dbp:INFT dbpo:coach ?c2.
    FILTER (?c1!=?c2)} } UNION
{GRAPH <http://gize.org/v2> { dbp:INFT dbpo:name ?p.
    dbp:INFT dbpo:coach ?c2.
    FILTER (?c1!=?c2) } } UNION
{GRAPH <http://gize.org/v1> { dbp:INFT dbpo:name ?p.
    dbp:INFT dbpo:coach ?c2.
    FILTER (?c1!=?c2)} }  }
```

The SPARQL query on the right is automatically generated and can be evaluated on existing processors. Note that the translation (because of the semantics of `PAST` described in Table 1) requires to look into all versions.

Example 2. Find the name of the coach of the Italian national football team after the sacking of Cesare Prandelli.

SPARQ–LTL	**Translation into SPARQL**

`SELECT ?n WHERE {`
`PAST {`
`dbp:INFT dbpo:coach dbp:CP.`
`NEXT {`
`dbp:INFT dbpo:coach ?n.`
`FILTER (?n != dbp:CP )`
`}`
`}`
`}`
	`SELECT ?n WHERE {`
	`{GRAPH <http://gize.org/v5> {`
	`dbp:INFT dbpo:coach dbp:CP.`
	`GRAPH <http://gize.org//v6>`
	`{dbp:INFT dbpo:coach ?n. FILTER (?n != dbp:CP )}`
	`} } UNION ..... UNION`
	`{GRAPH <http://gize.org//v1> {`
	`dbp:INFT dbpo:coach dbp:CP.`
	`GRAPH <http://gize.org//v2>`
	`{dbp:INFT dbpo:coach ?n. FILTER (?n != dbp:CP )`
	`} } }}`

In the previous query, `dbp:CP` is a shorthand for `dbp:Cesare_Prandelli`. As before, the translation of `PAST` makes usage of `UNION` queries over each versions v_i; then, for each v_i, `NEXT` checks in version v_{i+1} (via a `FILTER`) that the coach changed.

3 Conclusions

We have outlined Gize, which enables to set-up an infrastructure for querying historical RDF data on existing SPARQL processors. Gize adopts a simple approach to store different versions of the data and a powerful temporal extension of SPARQL called SPARQ–LTL. As a future work, we are considering approaches like RDF HDT [3] to improve the storage space consumption.

References

1. S. Borgwardt, M. Lippmann, and V. Thost. Temporalizing Rewritable Query Languages Over Knowledge Bases. *JWS*, 33:50–70, 2015.
2. G. D. De Giacomo and M. Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 2013.
3. J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary RDF Representation for Publication and Exchange. *JWS*, 19:22–41, 2013.
4. J. D. Fernández, P. Schneider, and J. Umbrich. The DBpedia Wayback Machine. In *SEMANTICS*, pages 192–195, 2015.
5. S. Gao, J. Gu, and C. Zaniolo. RDF-TX: A Fast, User-Friendly System for Querying the History of RDF Knowledge Bases. In *EDBT*, pages 269–280, 2016.
6. R. Mateescu, S. Meriot, and S. Rampacek. Extending SPARQL with Temporal Logic. Report, 2009.
7. H. Van de Sompel, R. Sanderson, M. L. Nelson, L. L. Balakireva, H. Shankar, and S. Ainsworth. An HTTP-based Versioning Mechanism for Linked Data. 2010.