Linking Historical Data on the Web*

Valeria Fionda[1], Giovanni Grasso[2]

[1] Department of Mathematics, University of Calabria, Italy
fionda@mat.unical.it
[2] Department of Computer Science, Oxford University, UK
giovanni.grasso@cs.ox.ac.uk

Abstract. Linked Data today available on the Web mostly represent snapshots at particular points in time. The temporal aspect of data is mostly taken into account only by adding and removing triples to keep datasets up-to-date, thus neglecting the importance to keep track of the evolution of data over time. To overcome this limitation, we introduce the LinkHisData framework to automatize the creation and publication of linked historical data extracted from the Deep Web.

1 Introduction

Most of the linked datasets today available on the Web offer a static view over the data they provide ignoring their evolution over time. Indeed, the temporal aspect is only considered by adding and removing information to keep datasets up-to-date [4]. However, while some information is intrinsically static because universally valid (e.g., the author of Harry Potter books is J. K. Rowling), a large portion of data are only valid from a particular point in time on (e.g., the first Harry Potter novel is available in Braille edition from May 1998) or are valid for a certain period (e.g., the price of the children paperback edition of the first Harry Potter book was £3.85 between the 10th and the 15th May 2014).

The possibility to include the temporal validity of RDF data opens the doors to the creation of new datasets in several domains by keeping track of the evolution of information over time (e.g., books prices, currency exchange rate) and publishing it as linked historical data. In turn, the availability of historical datasets would stimulate and enable a large number of applications (e.g., data analytics) and research areas [1]. Unfortunately, a recent work [6] showed that the amount of linked data with temporal information available on the Web is very small and to the best of our knowledge there are no proposals aimed at publishing historical dataset on the Web of Linked Data. In this paper we propose LinkHisData, a configurable framework to automatize the creation and publication of linked historical data extracted from the Deep Web. In particular, the focus is on extracting and making persistent transient information (e.g., the price of a book on a certain day) before becoming no longer accessible.

* V. Fionda was supported by the European Commission, the European Social Fund and the Calabria region. G. Grasso was supported by the European Commission's Seventh Framework Programme (FP7/2007–2013) from ERC grant agreement DIADEM, no. 246858.

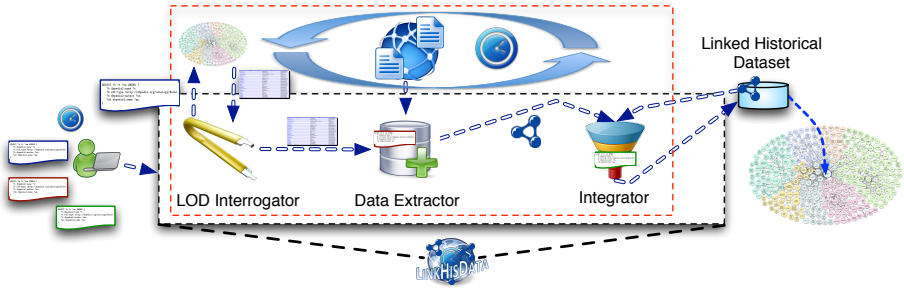

Fig. 1: An overview of the components of the LinkHisData framework.

2 The LinkHisData framework

LinkHisData (Linked Historical Data) is a configurable framework that builds on well-established semantic technologies and tools (e.g., SPARQL, RDF), as well as languages for Deep Web data extraction that have already been successfully employed by the LOD community (i.e., OXPath [3, 5]).

Fig. 1 shows the LinkHisData architecture. Its main components are: the LOD Interrogator, the Data Extractor, the Integrator, and the Linked Historical Dataset (LHD). The LOD Interrogator uses the SPARQL query and the endpoint address provided in input by the user to retrieve from the Web of Linked Data entities' URIs and related information. These data feed the Data Extractor that runs the OXPath wrapper, again provided in input by the user, to extract transient information from the Deep Web directly into RDF. OXPath is a modern wrapping language able to execute actions (e.g., click, form filling) and schedule periodic extraction tasks. The query and the wrapper may share variable names so that the wrapper is instantiate with the actual values provided by the LOD Interrogator. The RDF data generated by the Data Extractor populates LHD and are published on the Web. A single execution of the extraction and publication process produces RDF data with temporal information that represent a snapshot at the time of extraction. To produce historical data, the whole process is repeated at the frequency set by the user and for each repetition the Integrator is responsible for the integration of the fresh triples (with temporal validity set to the time of extraction) with the data already stored in LHD (whose validity dates back to a prior extraction) by executing the (set of) SPARQL query provided in input. Different inputs supplied by the user configure LinkHisData to extract and publish historical datasets in different domains.

3 LinkHisData: Book Price Example

We instantiate the LinkHisData framework for the extraction and publication of linked historical data about books prices extracted from Barnes&Noble (`www.bn.com`). The next query retrieves from DBpedia books and relative attributes[3]:

```
SELECT ?b ?t ?ab ?an WHERE {
  ?b dbp:name ?t. ?b rdf:type dbo:Book. ?b dbp:author ?ab. ?ab foaf:name ?an. }
```

[3] The prefixes used in the paper are taken from `www.prefix.cc`

```
doc("www.bn.com")//*[@id='keyword']/{?t ?an/}//*[@id='quick-search']/{"Books"/}
//*[@id='quick-search-1']//button/{click /}//li#search-result
[jarowrinkler(./li#title, ?t)=1][jarowrinkler(./li#auth, ?an)>.7]/{click /}/
/html:<(schema:Book(isbn))> [.: <owl:sameAs(?b)>]
 [.//*[starts-with(.,"ISBN")][1]/text()[2]: <schema:isbn=string(.)>]
 [.:<schema:author(schema:Person(?authorName))> [.: <owl:sameAs(?a)>]]
 [.: <schema:offers(schema:Offer lhd:HistoricalEntity)>
 [? .//::*[@itemprop="price"][1]: <schema:price=substring(.,2)> ]
 [? .//::*[@itemprop="price"][1]: <schema:priceCurrency=substring(.,1,1)> ]
 [? .:<schema:validFrom = now()> ]
```

Fig. 2: OXPATH RDF wrapper

This query returns a set of tuples containing the URI of a book (`?b`), its title (`?t`), the URI of its author (`?ab`) and the author's name (`?an`). For instance, for the book "The Firm" of J. Grisham the retrieved tuple is ⟨`dbpedia:The_Firm_(novel)`, "`The Firm`", `dbpedia:John_Grisham`, "`John Grisham`"⟩. The remainder of the example uses the values of this tuple to instantiate the various components.

The tuples returned by the LOD Interrogator constitute the input for the Data Extractor which runs the OXPath wrapper shown in Figure 2 (for space reasons some parts are simplified or omitted). Here, the variables (e.g., `?b`) refer to the corresponding ones in the LOD Interrogator SPARQL query. We assume some familiarity with XPath to illustrate the wrapper. It comprises three parts: *(i)* navigation to the pages containing relevant data, *(ii)* identification of data to extract, and *(iii)* RDF output production. In our example, we use types and properties from `schema.org` (e.g., `Book`, `Offer`, `price`). For instance, for our example tuple about "The Firm", the wrapper produces the following RDF output:

```
lhd:b9780440245926 a schema:Book ;  owl:sameAs dbpedia:The_Firm_(novel);
 schema:isbn "9780440245926"; schema:author lhd:John_Grisham;
 schema:offers lhd:off_123.
lhd:off_123 a schema:Offer,lhd:HistoricalEntity ; schema:price "9.21";
 schema:priceCurrency "$"; schema:validFrom "2014-06-28".
lhd:John_Grisham a schema:Person ; owl:sameAs dbpedia:John_Grisham.
```

The wrapper encodes part *(i)* in lines 1–2. Firstly, the website is loaded, then, the search form is filled with ("`The Firm John Grisham`") i.e., book title and author name, plus the search is restricted to the book category. Finally, the submit button is clicked and all the books on the result page are selected by the expression `//li.search-result`. However, many of the results may not refer to the book of interest (e.g., also collections/sets of books containing it are retrieved) and it is absolutely crucial to identify the correct entities as these will be linked back to the original entities in DBpedia. We address this problem (part *(ii)*) by using the Jaro-Winkler distance [2] to match author name and book title. This metric has been widely used in record linkage and performs particularly well on person and entity's names. Our wrapper (line 3) demands for a perfect matching on the title and a significant similarity (>0.7) on the author's name to deal with different variations (e.g., "J. Grisham" or "John Ray Grisham, Jr").

This strategy prevents our framework from overtaxing the site by extracting (a possibly huge quantity of) data on non-interesting books, that would be only later discarded by a costly post-processing linking phase. For example, for "The Firm" we correctly identify only 2 books out of the original 22 results.

Part *(iii)* is realized by visiting the detail page of each selected book via the action `{click/}`. An *RDF extraction marker* is used to create a `schema:Book` instance (line 4) and by explicitly linking it to the corresponding DBpedia URI via `owl:sameAs`. The unique URI for the created book instance relies on the functional dependency to its `isbn` (`schema:Book(isbn)`). This ensures that this URI will be always the same within the subsequent extractions, allowing to refer to the right entity in the integration phase. The wrapper also creates one linked data entity for the book author (line 6) and one for the offer (line 7), respectively. The former is of type `schema:Person`; its URI is created on the basis of the author name (*?an*) and is linked via `owl:sameAs` to the author on DBpedia. The latter is of type `schema:Offer` and `lhd:HistoricalEntity`, type used to mark entities with temporal validity. Its URI is randomly generated to ensure different URIs for consecutive extractions. Some properties of the offer are also extracted (e.g., the `schema:price`) and the `schema:validityFrom` is set to `now()`, the current date.

The Integrator takes in input, for each book (`?b`), the set of triples `T` as produced by the Data Extractor and it is responsible for integrating them with those already present in LHD. In particular, it deals with entities having a temporal validity, e.g., book offers and their prices in our example. Each book in LHD may have associated several offers that represent the evolution of the price over time. However, only one of them provides the current selling price (i.e., the one not having a `schema:validThrough` triple). Therefore, for each book (`?b`), the Integrator instantiates the query template (provided by the user) to retrieve such current price (`?p`) and its corresponding offer (`?o`) from the historical dataset. If `?b` is not already present in the historical dataset, the triples in `T` are added to it. Otherwise, the Integrator compares the current price (`?p`) with the price of the offer in `T` (freshly extracted). If they differ, the price validity is updated by adding to the dataset both the triple (`?o`, `schema:validThrough`, `now()`) and the offer in `T`. Together they provide a new piece of historical information.

References

1. O. Alonso, J. Strötgen, R. A. Baeza-Yates, and M. Gertz. Temporal information retrieval: Challenges and opportunities. In *TWAW*, volume 813, pages 1–8, 2011.
2. W.W. Cohen, P.D. Ravikumar, and S.E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.
3. T. Furche, G. Gottlob, G. Grasso, C. Schallhart, and A.J. Sellers. OXPath: A language for scalable data extraction, automation, and crawling on the deep web. *VLDB J.*, 22(1):47–72, 2013.
4. T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, and A. Hogan. Observing linked data dynamics. In *ESWC*, pages 213–227, 2013.
5. J. Lehmann, T. Furche, G. Grasso, A.C. Ngonga Ngomo, C. Schallhart, and C. et al. Unger. deqa: Deep web extraction for question answering. In *ISWC*, 2012.
6. A. Rula, M. Palmonari, A. Harth, S. Stadtmüller, and A. Maurino. On the diversity and availability of temporal information in linked open data. In *ISWC*, 2012.