Full-text Support for Publish/Subscribe Ontology Systems

Lefteris Zervakis[1], Christos Tryfonopoulos[1], Antonios Papadakis-Pesaresi[2], Manolis Koubarakis[2], and Spiros Skiadopoulos[1]

[1] Dept. of Computer Science and Technology, University of Peloponnese
[2] Dept. of Informatics and Telecommunications, University of Athens

Abstract. We envision a publish/subscribe ontology system that is able to index millions of user subscriptions and filter them against ontology data that arrive in a streaming fashion. In this work, we propose a *SPARQL extension* appropriate for a publish/subscribe setting; our extension builds on the natural semantic graph matching of the language and supports the creation of *full-text subscriptions*. Subsequently, we propose a main-memory *subscription indexing algorithm* which performs both semantic and full-text matching at low complexity and minimal filtering time. Thus, when ontology data are published matching subscriptions are identified and notifications are forwarded to users.

System overview

Resource Description Framework (RDF) constitutes a conceptual model and a formal language for representing resources in the Semantic Web. It is also the data format of choice for modern *publish-subscribe* ontology systems, which demand sophisticated data representation and efficient filtering mechanisms to match massive ontology data against millions of *user subscriptions* (also referred to as *continuous queries*). The SPARQL query language is currently the W3C recommendation for querying the Semantic Web. The graph model over which it operates naturally joins data together and represents a fully-fledged language. However, it still lacks the support of a complete *full-text retrieval* mechanism, beyond existing regular expression support, with sophisticated algorithms and data structures to minimise processing and memory requirements.

In this work, we focus on full-text filtering of ontology data that contain RDF literals in their property elements. To preserve the expressivity of SPARQL, we view the full text operations as an additional *filter* of the subscription variables. In this context, we define a new binary operator *ftcontains* that takes a variable of the subscription and a full-text expression that operates on the values of this variable as parameters. An example of a SPARQL subscription with full-text support is shown below.

```
SELECT ?article
WHERE  {?publisher rdf:type      Publisher.
        ?publisher publishes     ?article.
        ?article   articleText   ?articleText.
FILTER  ftcontains (?articleText, "economic" ftand "crisis")}
```

We focus on RDF triples where the *subject* is always a node element and the *predicate* denotes the subject's relation to the *object*, which is a literal expressed as a typed or untyped string. A full text expression is evaluated only

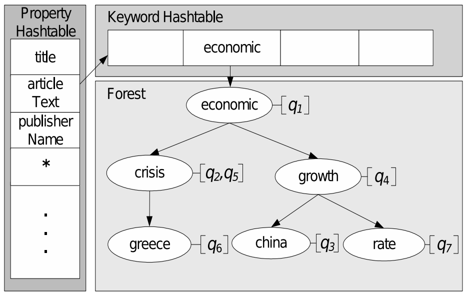

Fig. 1. Subscription indexing scheme

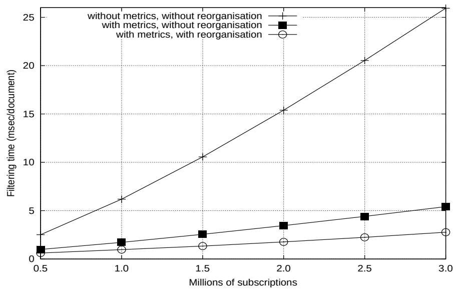

Fig. 2. Filtering time/document (msecs)

against a literal; thus the variable of the subscription can only be the object of a triple pattern. The expressions supported involve the usual *Boolean operators* (denoted by *ftand*, *ftor*, etc.), as well as *proximity* and *phrase* matching. Below we present an example of a full-text SPARQL subscription that will match all *rdf:type Article* node elements, with a property named *title* containing a string literal with the keywords *"economic"* and *"crisis"*.

To perform the semantic matching, we define a Semantic Match Table in the spirit of [1], where a two-level hash table is used to represent the series of joins in a SPARQL subscription as a connected chain. We extend this idea to provide a hashing scheme that is able to accommodate all possible types of triple patterns in SPARQL subscriptions. Additionally, to support the full-text features introduced in the SPARQL subscriptions, we utilise a *property hash table* that uses as key the constant part of the triple pattern in the SPARQL subscription. This hash table provides access to a data structure, which comprises of (i) *tries* storing the keywords contained in the full-text part of subscriptions and (ii) a *keyword hash table* that allows fast access to the trie roots. Figure 1 shows these data structures for a set of seven user subscriptions.

User subscriptions are organised into tries extending the approach of [2] to rely on *common subsets* of subscriptions. The main idea behind the indexing algorithm is to use tries to capture common elements of subscriptions. To do so, we utilise *metrics* to locate the best possible indexing position in the forest of tries. Since our algorithm is influenced by the order of insertion of subscriptions (due to greedy subscription indexing), a statistics-based *subscription reorganisation* is employed. In the reorganisation phase of the algorithm, a scoring mechanism is utilised to modify the order of subscription indexing for all subscriptions inserted since the last reorganisation of the forest. In our evaluation we used $3.1M$ extended abstracts downloaded from DBpedia as incoming RDF documents and artificially generated subscription databases of varying sizes. Figure 2 shows the filtering time when (i) no metrics for the best indexing position in the forest are employed (deterministic subscription indexing), (ii) metrics are employed, but no re-organisation is used, and (iii) both metrics and reorganisation are employed.

References

1. Park, M.J., Chung, C.W.: ibroker: An intelligent broker for ontology based publish/subscribe systems. In ICDE 2009.
2. Tryfonopoulos, C., Koubarakis, M., Drougas, Y.: Information filtering and query indexing for an information retrieval model. In ACM TOIS 2009.