**Exelixis: Evolving Ontology-Based Data Integration System**

Haridimos Kondylakis  
FORTH-ICS  
kondylak@ics.forth.gr

**ABSTRACT**

The evolution of ontologies is an undisputed necessity in ontology-based data integration. Yet, few research efforts have focused on addressing the need to reflect ontology evolution onto the underlying data integration systems. We present *Exelixis*, a web platform that enables query answering over evolving ontologies without mapping redefinition. This is achieved by rewriting queries among ontology versions. First, changes between ontologies are automatically detected and described using a high level language of changes. Those changes are interpreted as sound global-as-view (GAV) mappings. Then query expansion is applied in order to consider constraints from the ontology and unfolding to apply the GAV mappings. Whenever equivalent rewritings cannot be produced we a) guide query redefinition and/or b) provide the best “over-approximations”, i.e. the minimally-containing and minimally-generalized rewritings. For the demonstration we will use four versions of the CIDOC-CRM ontology and real user queries to show the functionality of the system. Then we will allow conference participants to directly interact with the system to test its capabilities.

**Categories and Subject Descriptors**  
H.2.5 [Heterogeneous Databases]: Program Translation.

**General Terms**  
Algorithms, Languages, Theory.

**Keywords**  
Ontologies, Data Integration, Query Rewriting

1. **INTRODUCTION**

The development of new scientific techniques and the emergence of high throughput tools have led to a new information revolution. The amount, diversity, and heterogeneity of the information available have led to the adoption of data integration systems in order to manage it and further process it. However, the integration of several data sources raises many semantic heterogeneity problems.

By accepting an ontology as a point of common reference, naming conflicts are eliminated and semantic conflicts are reduced. Ontologies are used to identify and resolve heterogeneity problems, usually at schema level, as a means for establishing an explicit formal vocabulary to share. During the last years, ontologies have been used in database integration, obtaining promising results, for example in the fields of biomedicine and bioinformatics. When using ontologies to integrate data, one is required to produce mappings, to link similar concepts or relationships from the ontology/ies to the sources by way of an equivalence. This is the mapping definition process [1] and the output of this task is the mapping, i.e. a collection of mappings rules. In practice, this process is done manually with the help of graphical user interfaces and it is a time-consuming, labour-intensive and error-prone activity. Defining the mappings between the ontology and the schemata is not a goal in itself. The resulting mappings are used for various integration tasks such as data transformation and query answering.

Despite the great amount of work done in ontology-based data integration, an important problem that most of the systems tend to ignore is that ontologies are living artifacts and subject to change [2]. Due to the rapid development of research, ontologies are frequently changed to depict the new knowledge that is acquired. The problem that occurs is the following: when ontologies change, the mappings may become invalid and should somehow be updated or adapted. A typical solution would be to regenerate the mappings and then regenerate the dependent artifacts each time the ontology evolves. However, as this evolution might happen too often, the overhead of redefining the mappings each time is significant.

The approach, to recreate mappings from scratch each time the ontology evolves, is widely recognized to be problematic, and instead, previously captured information should be reused. A nice overview of approaches trying to tackle similar problems can be found on [3] and [4]. The most relevant approaches that could be employed for resolving the problem of data integration with evolving ontologies is mapping adaptation [5] and mapping composition and inversion [6, 7] where mappings are changed automatically each time the ontology evolves. However, in mapping adaptation there is no guarantee that after repeatedly applying the algorithm, the semantics of the resulting mappings will be the desired ones and in mapping composition and inversion complex language of mappings should be employed (second-order dependencies) where the inversion is not guaranteed [6].

The lack of an ideal approach led us to design and develop *Exelixis* for evolving RDF/S ontologies, by harvesting recent theoretical advances on the areas of ontology change [8] and query rewriting [9]. In our case the initial mappings remain unaffected while query rewriting is employed for assembling a coherent view of the world out of each specific setting.

By using our JQuery-based web interface, users are able to register different RDF/S ontology versions that will be used as global schemata by the underlying ontology-based data integration systems. The system automatically detects and stores the sequence of changes between these versions in order to be later used on the query rewriting phase. The sequences of changes produced are unique, complete and non-ambiguous as shown on
and their composition and inversion is always possible to be computed [4]. Then the user is able to choose an ontology version, to visualize it using current state-of-the-art web ontology viewers, and to formulate queries. Queries to our system are posed in terms of one ontology version using the SPARQL query language. Then, query rewriting is employed in order to rewrite queries among ontology versions and assemble the results from all data integration systems that might use different ontology version as global schema.

Query rewriting is performed in two steps, namely: a) query expansion and b) valid rewriting as shown in Figure 1. In query expansion the query is rewritten in order to consider constraints from the ontology version (subClass and subProperty transitivity constraints). In the second phase, the sequences of changes among ontologies are interpreted as sound GAV mappings and unfolding is used in order to rewrite queries between ontology versions. The rewritten queries are forwarded to the underlying data integration systems and the answers are returned to the user.

(ii) The second option we provide to our users is the OWLSight ontology browser offered by the Pellet reasoner. The interface is close to the current state-of-the-art ontology editors and it is more intuitive than the jOWL approach. However, it is not embedded on our web site (it opens in another web page).

(iii) The last visualization option we provide is the Starlion tool, which provides top-k and force-directed diagrams for navigating through the ontology.

Besides ontology viewers the user is able to graphically formulate a SPARQL query using a modified version of the NITELIGHT plug-in. The users can drag-n-drop variables, classes and properties from the selected ontology and design the graph pattern of the corresponding SPARQL query.

Finally, the underlying ontology-based data integration systems selected, are instances of the MASTRO system. MASTRO is a state-of-the-art data integration system that uses GAV mappings to relate an ontology to relational sources.

![Figure 1. Query Rewriting](http://example.com/figure1.png)

Despite the fact that query rewriting always terminates, the queries issued to other ontology versions might fail. This is because non-information preserving changes may have occurred (such as the deletion of a queried class). To tackle this problem, we propose two solutions: a) either to provide approximate rewritings by means of minimally-containing or minimally-generalized queries, or b) to provide insights for the failure by means of change paths, thus driving query redefinition. More information on the aforementioned techniques, formal definitions, proofs, and an evaluation of the system can be found on [4]. Moreover, a first prototype of the Demo is online. A richer and more stable version of the interface is about to be released.

2. SYSTEM ARCHITECTURE

The architecture of the system and its components are shown in Figure 2. At first, the user should be able to visualize the ontology in order to formulate queries. To that purpose we offer three options to the user:

(i) The first one is by embedding the ontology hierarchy directly on our web page. This is achieved using the OWL API. By using the specific API we offer to the user the option to search for a class or a property and to visualize the corresponding description.

(ii) The second option we provide to our users is the OWLSight ontology browser offered by the Pellet reasoner. The interface is close to the current state-of-the-art ontology editors and it is more intuitive than the jOWL approach. However, it is not embedded on our web site (it opens in another web page).

(iii) The last visualization option we provide is the Starlion tool, which provides top-k and force-directed diagrams for navigating through the ontology.

Besides ontology viewers the user is able to graphically formulate a SPARQL query using a modified version of the NITELIGHT plug-in. The users can drag-n-drop variables, classes and properties from the selected ontology and design the graph pattern of the corresponding SPARQL query.

Finally, the underlying ontology-based data integration systems selected, are instances of the MASTRO system. MASTRO is a state-of-the-art data integration system that uses GAV mappings to relate an ontology to relational sources.

![Figure 2. Platform Architecture](http://example.com/figure2.png)

2.1 Core Components

The core components of our system are the a) Expander, the b) Valid Rewriter and the c) the Change Path Generator.

The Expander uses the Quonto reasoner in order to automatically identify the subClass and subProperty constraints from the ontology. Having identified those constraints the Expander, rewrites the query according to them. For example, if we query for the “virtual instances” of a class Person, and the class Actor is a subClass of Person, the query will be rewritten to ask for instances of both Person and Actor.

Then, the Valid Rewriter uses the GAV mappings, constructed earlier, to rewrite queries among ontology versions. For example if in one version the class Actor has been renamed to the class Agent the query that previously asked for instances of Actor will now be rewritten to ask for instances of Agent.

However, it might be the case that the class Actor in an ontology version no longer exists since it might has been deleted. So, over approximations should be provided. For example we could search for the parent of the deleted class and ask for that class instead if

---

1. [http://139.91.183.29:8080/exelisis/](http://139.91.183.29:8080/exelisis/)
it is not deleted in the next ontology version. The query produced is the minimally-generalized rewriting of q. To identify the parent of a class/property the Pellet reasoner is used. Moreover, if a conjunct of the query no longer exists in the target ontology version, it can be removed (under the proper conditions), thus providing another aspect of over-approximation, i.e. the minimally-containing rewriting.

Finally, the user is able to resolve the evolution of the ontology only for a specific part of the ontology (for example the part that invalidates the query). This is done by calculating the change paths for a specific class, property or a change operation.

3. DEMONSTRATION

To demonstrate the functionalities of the Exelixis platform, we will use 4 versions of the CIDOC-CRM [10] ontology. CIDOC-CRM is an ISO standard which consists of nearly 80 classes and 250 properties. The demonstration will proceed in five phases and parts of the involved interface is shown in Figure 3.

Figure 3. The different stages of query rewriting

(i) Configuration: The demonstration will start by presenting the different ontology versions using our web ontology viewers. Then, two instances of the MASTRO ontology-based data integration system will be registered in our platform, each one of them using a different ontology version as global schema.

(ii) High-level changes: In this phase representative cases from the total 711 identified changes among the four ontology versions will be presented and explained. For those changes, the corresponding GAV mappings will be described as well. Moreover, the inverse of a sequence of changes will be presented and explained.

(iii) Visual Query formulation: Then, a SPARQL query will be formulated using our Visual SPARQL query generator. The query will be constructed in terms of the latest ontology version.

(iv) Query Rewriting: The query will be issued to the system and the results as well as the rewriting steps will be presented to the conference participants. Initially, the expanded query will be shown. Then the system will identify the ontology versions of the two underlying data integration systems and will try to rewrite the input query to the ontology versions they use. The specific change operations that are used in each case will be shown and the resulted query as well. Finally the answers of the two data integration systems will be provided.

(v) Change Paths & Approximations: In this phase queries will be presented for which no equivalent rewriting exists. So the minimally-contained and the minimally-generalized rewritings will be presented and explained. For the changes that lead to query invalidation, the change path will be presented, describing the evolution for that specific part of the ontology.

(vi) “Hands-on” phase: In this phase conference participants would be invited to directly interact with the system, either by issuing already stored pragmatic queries, or by formulating new queries at will.

4. CONCLUSION

In this demonstration, we present a new platform that enables query answering over evolving ontology-based data integration systems. This is achieved by modelling ontology evolution using a high-level language of changes and then rewriting queries among ontology versions. Whenever, no equivalent rewriting exists, the identification of failures is presented to the user, and best over-approximations are provided as well. We demonstrate our platform using pragmatic data sets, and allow the direct participation of the conference participants to test its capabilities. To the best of our knowledge, no other system today is capable of automatically answering queries over multiple ontology versions.

5. REFERENCES


