Incremental, Per-Query Ontology Matching with RepMine

Thomas Kowark, Keven Richly, Matthias Uflacker, Hasso Plattner
Hasso Plattner Institute
August-Bebel-Str 88
Potsdam, Germany
{firstname.lastname}@hpi.de

ABSTRACT

Ontology matching enables applications, such as automated data transformation or query rewriting. As it requires domain knowledge, it needs to be carried out by expert users, whose time is scarce and, therefore, should be used efficiently. To this end, the RepMine system presented in this paper does not treat ontology matching as a task of its own, but integrates it into a semi-automated query translation process. By that, users perform a task with immediate benefit for them and simultaneously contribute to alignments between ontologies. Furthermore, the overall task of matching two ontologies is split on a per-query basis and, thus, can be performed incrementally by all system users.

Keywords

Ontology Matching, User Involvement, Query Translation

1. INTRODUCTION

In the context of the Semantic Web, ontology matching is a prerequisite for applications such as automated data transformation or query rewriting [7]. It is necessary, since for some domains, multiple ontologies exist that describe the same real world entities using different terminology and semantics. The conference ontologies of the Ontology Alignment Evaluation Initiative[1] (OAEI) are an example for that. Other domains, such as biomedicine, are too complex to be captured in a single, comprehensive ontology, hence, different aspects are formalized in different ontologies.

The main goal of ontology matching is then to detect correspondences, i.e., determine how the same concepts are expressed within different ontologies. While this task can be automated to some extent, recent OAEI campaigns have shown that automatic ontology matching tools are not yet capable of consistently producing complete and correct alignments (collections of correspondences) without the input of expert users, whose time is a scarce resource. Therefore, interactive ontology matching tools aim to limit the amount and extent of user interactions, as much as possible [7].

The tool presented in this paper takes a different approach. Instead of optimizing the user interaction with a tool specifically designed to match ontologies, we focus on a task with *immediate* benefit for the end user and collect correspondences between ontology elements as a byproduct. In particular, we exploit the task of query rewriting, that is, translating a query that was initially written on one ontology in order to be executable on another ontology. Through that approach, users remain within the tool that performs their desired task, yet still contribute to an incrementally evolving ontology alignment. Furthermore, each contributed correspondence is immediately available for all system users and will consequently be reviewed, corrected, or extended.

The paper outlines the standard use case we cover with our tool, explains its implementation, and briefly discusses related approaches. To demonstrate the tool's capabilities, a screencast is provided[2].

2. APPLICATION WORKFLOW

RepMine was initially created in the context of software **rep**ository **min**ing. Hence, its target user groups are researchers in this area who want to perform analyses of software development data on multiple datasets (e.g., stemming from different projects for validation purposes), without having to manually perform data transformation or query rewriting. Abstracting from this initial use case, every user that wants a query to be answered from different data sources is a target user of our system.

The overall concept of our system is shown in Figure 2. Regardless of the use case, RDF ontologies that describe the source and target data store are the foundation for our approach. These ontologies can either be provided as-is, e.g. in case of RDF triple stores, or need to be extracted. Section 3 provides details about ontology creation process. Once the ontology is present, queries can be formulated.

RepMine provides a graphical abstraction for query formulation and thereby follows recent, comparable approaches that aim to provide data analytics without requiring knowledge about query languages such as SPARQL [8]. Figure 2 shows an example of the notation. Contrary to string based representations, it provides an explicit link to ontology concepts used within the query and, hence, simplifies determining which query parts have to be translated in which manner.

[1] http://oaei.ontologymatching.org

[2] https://youtu.be/-6iyrXRmXQU

WWW'16 Companion, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4144-8/16/04.
http://dx.doi.org/10.1145/2872518.2890554.

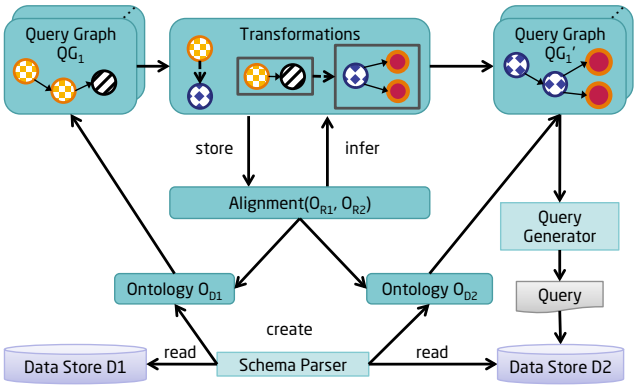

Figure 1: Schematic overview of the main entities within the RepMine workflow. Query graphs based on an input ontology ($\mathcal{O}_{D1}$) are transformed to semantically equivalent output graphs based on an output ontology ($\mathcal{O}_{D2}$). Throughout the transformation process, transformation steps are stored as correspondences within the alignment between the ontologies. Existing correspondences are used to infer graph transformations to build the output graph. Finally, queries suitable for the target data store are created and executed.

The query graphs are created either automatically through parsing input queries, or manually through a graphical editor. Either way, the main task for users that need this query to be executed on a different ontology is to produce an equivalent output graph. Equivalent in that regard refers to a graph that denotes a query which would yield identical results if the data from the input ontology was to be expressed in terms of the target ontology.

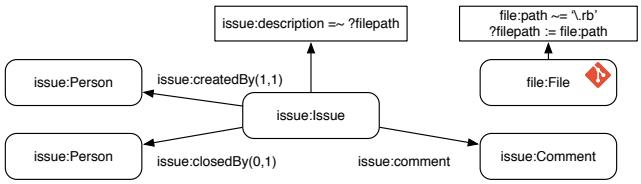

Figure 2: Query graph providing an abstraction for a query that retrieves comments on issues that are linked to at least one file. Rounded nodes depict instances of classes, squared nodes depict datatype properties, and object properties are expressed through arrows between rounded nodes. Nodes can use concepts from different ontologies (in this case, "file" and "issue") to express federated queries. Graph elements can be used as variables through using the "?" notation known from SPARQL.

2.1 Ontology Matching

The creation of this output graph comprises three steps, which are repeated until correspondences for each of the input elements are provided and the user considers the output graph to be equivalent to the input:

- **#1:** Automatic ontology matching tools, such as Agreement Maker Light (AML) [2] or LogMap [4] are run to detect simple correspondences between ontology elements (e.g., detecting that two ontologies contain a class called "Issue"). Based on the found correspondences nodes in the output graph are automatically created by RepMine.

- **#2:** Users complete the output graph by adding further nodes, relations, or attributes. For each added element, users are asked to define how they relate to elements of the input graph. This can either include simple correspondences (a node being substituted by another node) or complex correspondences [5] (e.g., a "Bug" in the input ontology is equivalent to an "Issue" in the output ontology that has an attribute called "type" with a value "bug").

- **#3:** After each provided correspondence, the automated ontology matching tools are run again with the added information in order to detect potential new correspondences.

Users interact with the editor shown in Figure 3. It is connected to the source and target ontologies to guide users in the creation of query graphs. For example, if two nodes are connected by a relation, only object properties defined between the two selected classes can be selected as the relation type. Once the output graph is complete, the system is able to translate it into a query suitable for the desired target datastore. Currently, RepMine supports SPARQL endpoints, Neo4j[3] property graphs, and relational databases.

2.2 Correspondence Validation & Extension

Only looking at single queries, the system allows users to translate them in a graphical manner and with the help of automatic ontology matching tools to a representation suitable for their target data store of choice. Its full potential, however, comes into effect by storing queries as part of a repository. Each user can add new queries or build on existing ones by creating clones that can be extended or simplified, as needed. Each concept used within a query only needs to be translated once to a target ontology, and the associated correspondence is immediately available for all other queries, as well.

For one, this approach allows for constant validation of provided correspondences. If a user detects that a concept is wrongly translated, they can create an alternative to the existing correspondences and the creator of the initial one gets notified. Now the two users can discuss which of the provided correspondences is correct and shall be used in the future. Also, a newly added correspondence could lead to inconsistencies in the generated alignment[4], which need to be solved by either adapting the correspondences or the target ontology, if necessary.

Besides contradicting each other, correspondences can also extend existing ones. Consider the case of a "Bug" class in the original ontology. One user translates it to a target ontology through an "Issue" node with a property "type" set to "bug". Another user determines that, in their dataset,

[3]https://neo4j.com

[4]Consider a case where two classes and two datatype properties are declared equivalent but in the target ontology the class is excluded from the domain of the property

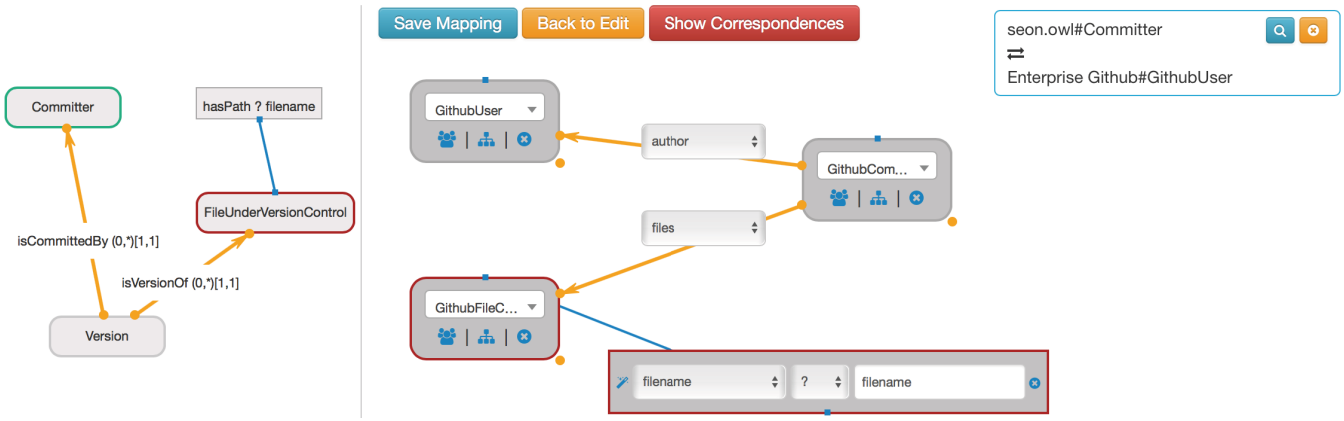

Figure 3: Transformation of an input graph element (marked red) to an output element on the right hand side. Already matched elements are marked green. Users are automatically guided to the next unmatched element until the entire graph is translated. Existing correspondences can be highlighted through an overview on the right side of the editor.

the type field is not always set correctly but instead the "title" property contains the word "bug". Hence, they extend the existing correspondence by adding the respective attribute constraint node in the output graph and linking it with the existing attribute constraints through an "OR" operator. The system accordingly detects that the new correspondence extends the existing one, stores only the new one, and also notifies the creator of the original correspondence so they can review the extension and discuss, if needed.

Finally, it might be possible, that no correspondence can be provided for an element, at all. For example, if the input ontology contains a property reflecting the sentiment of a comment, and such a value is not present in the target ontology, sentiment analysis would need to be performed. To cater to this issue, we implemented a feature that allows to specify "virtual elements" in the output graph, which link to service invocations (see Figure 4). These service invocations are performed prior to query execution and its results are stored in the target data store.

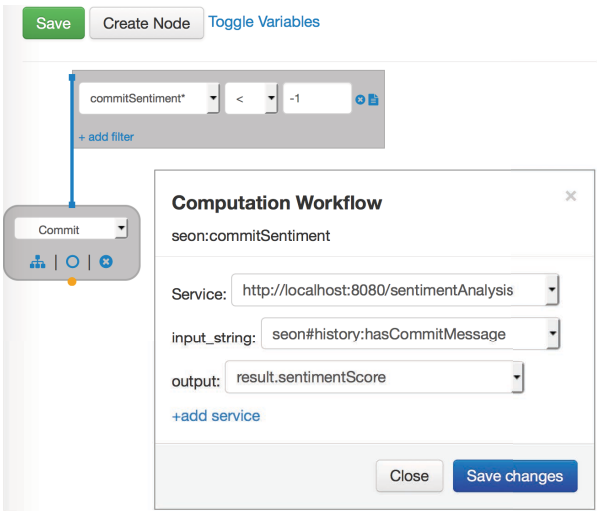

Figure 4: Specification of a virtual attribute calculation through a local sentiment analysis service.

2.3 Limitations

While the presented workflow guides users in providing correspondences between ontology elements, it does not foster the initial understanding of the target and source ontology. Systems, such as Optique [3] instantly provide result sets from target data stores during query creation in order to allow for a more explorative modelling. We plan on integrating such a feature in future versions of our tool.

Furthermore, the encoded queries do not contain aggregation operations (group by, count, etc.). This is due to the original application domain, which benefits from a separation of collaboration events (e.g., a user committing a file) and metrics (e.g., amount of commits per day, amount of commits on certain files by certain users, etc.), as events only need to be modelled once and can be reused in multiple metrics without having to duplicate modelling efforts (see Figure 5 for metric example). The two editors could be merged to allow modelling both queries and aggregations within a single view.

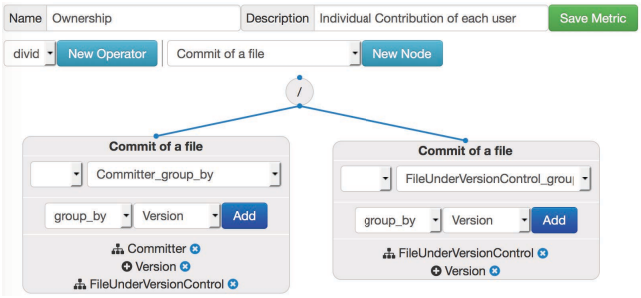

Figure 5: Creation of an ownership metric based on the query graph that represents a commit of a file by a user.

Creating large, complex queries currently requires the creation of accordingly large graphs as no means for linking existing query graphs is present. We plan on implementing this feature through representing existing graphs as special types of nodes, from which users can select the element which they want to use for connecting to the additional graph elements.

Once the proposed additions are implemented, the tool will be made available as an open-source project on Github.

3. IMPLEMENTATION

RepMine is implemented as a standalone web application written in Ruby on Rails. Its architecture is presented in Figure 6. To increase the extensibility of the platform, open-source, third party tools and libraries were used to implement required functionality, whenever possible. In addition to the automatic ontology matching systems, this includes libraries for extracting ontologies from relational databases[5] or MongoDB document stores[6]. Furthermore, query creation is only implemented in a custom fashion for SPARQL and Neo4j's Cypher language. SQL queries are generated by the ontop system [6] based on SPARQL.

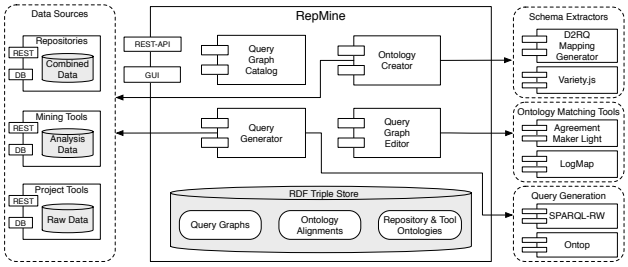

Figure 6: RepMine components and employed external tools and libraries.

The front end of the system is created using the bootstrap framework[7] for the general look and feel, as well as jsPlumb[8] for the graph editors. All created graphs and the generated correspondences are stored in an Allegro Graph Triple Store[9], which scales to billions of triples, and therefore is able to contain a multitude of query graphs, ontologies, and correspondences while still allowing for fluid user interaction. Currently, the system is deployed as one central application, but with small adoptions it could also read-from and write-to existing repositories for ontology alignments to foster reuse and reach of correspondences.

4. RELATED WORK

In the area of user supported ontology matching, state of the art tools, such as AML [2] or LogMap [4] provide an interactive mode that gathers user input to reason about potential correspondences ("Is X equivalent to Y?") and iteratively increase precision of the created alignment. Hence, these tools are primarily targeting administrators that provide alignments, which can then be used to perform the task desired by the end user of the main system. The Optique system [3], for example, implements this use case and thereby frees end users from performing ontology matching tasks, entirely. The downside of this approach, however, is that users cannot just add a new data source they would like to query, without having to use ontology matching tools, as well, in order to create alignments required for query rewriting. We therefore see our system as a supporting approach that could be employed to assist administrators by allowing users to contribute to alignments, and empower users to create alignments without their help, at all. A similar approach was presented by Ellis et al. with the Helix system [1], but, contrary to our system, they did not allow for specification of complex correspondences.

[5]The D2RQ Platform, http://d2rq.org
[6]variety.js, https://github.com/variety/variety
[7]http://getbootstrap.com
[8]https://jsplumbtoolkit.com
[9]http://franz.com/agraph/allegrograph/

5. SUMMARY

In this demonstration paper, we presented the RepMine system, which allows end users to translate graphical representations of SPARQL queries to different ontologies and provide ontology correspondences while doing so. Using these correspondences, the query translation effort incrementally decreases until a complete alignment is created. The tool is available on Github and distributed under an MIT license [10].

6. REFERENCES

[1] J. B. Ellis, O. Hassanzadeh, K. Srinivas, and M. J. Ward. Collective ontology alignment. In *Proc. 8th ISWC Workshop on Ontology Matching*, 2013.

[2] D. Faria, C. Pesquita, E. Santos, M. Palmonari, I. Cruz, and F. Couto. The AgreementMakerLight Ontology Matching System. In R. Meersman, H. Panetto, T. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. De Leenheer, and D. Dou, editors, *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, volume 8185 of *Lecture Notes in Computer Science*, pages 527–541. Springer, Berlin, Heidelberg, 2013.

[3] M. Giese, A. Soylu, G. Vega-Gorgojo, A. Waaler, P. Haase, E. Jimenez-Ruiz, D. Lanti, M. Rezk, G. Xiao, O. Ozcep, and R. Rosati. Optique: Zooming in on big data. *Computer*, 48(3):60–67, Mar 2015.

[4] E. Jiménez-Ruiz and B. C. Grau. Logmap: Logic-based and scalable ontology matching. In *International Semantic Web Conference (ISWC)*, volume 7031 of *Lecture Notes in Computer Science*, pages 273–288. Springer, October 2011.

[5] H. Paulheim, S. Hertling, and D. Ritze. Towards evaluating interactive ontology matching tools. In P. Cimiano, O. Corcho, V. Presutti, L. Hollink, and S. Rudolph, editors, *The Semantic Web: Semantics and Big Data*, volume 7882 of *Lecture Notes in Computer Science*, pages 31–45. Springer Berlin Heidelberg, 2013.

[6] M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-based data access: Ontop of databases. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 558–573, 2013.

[7] P. Shvaiko and J. Euzenat. Ontology matching: State of the art and future challenges. *IEEE Trans. on Knowl. and Data Eng.*, 25(1):158–176, Jan. 2013.

[8] A. Soylu, M. Giese, E. Jimenez-Ruiz, G. Vega-Gorgojo, and I. Horrocks. Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users. *Universal Access in the Information Society*, pages 1–24, 2015.

[10]http://github.com/hpi-epic/repmine