Ontology Reuse and Exploration via Interactive Graph Manipulation

Immanuel Normann[1] and Oliver Kutz[2]

[1] Department of Linguistics and Literature, University of Bremen, Germany
normann@uni-bremen.de
[2] Research Center on Spatial Cognition (SFB/TR 8), University of Bremen, Germany
okutz@informatik.uni-bremen.de

Abstract. Reusing ontologies can significantly accelerate the development of new ontologies, and therefore interoperability. However, reuse of ontologies calls for efficient means to identify reusable parts within and across existing ontologies. This paper presents a novel, graphical user interface to explore semantic overlaps between ontologies in order to identify parts for knowledge reuse. Existing tools typically only compare pairs of ontologies. Our approach allows for exploring concept mapping links across arbitrarily many ontologies, represented as so-called hyper-ontology graphs. To cope with the inherent complexity of such graphs, we allow for a variety of information hiding techniques tailored to specific exploration tasks. By manipulating the hyperontology graph interactively the user can refine the concept mapping network to his needs. Finally, we show how the resulting graph can be used to automatically synthesise a new ontology by extracting the corresponding modules and merging them appropriately.

Key words: Ontology repositories, modularity, matching, information visualisation

1 Introduction

Creating a high-quality ontology from scratch is a time consuming endeavour. Reusing existing ontologies or fragments of them is the obvious thing to do here. In most cases, the envisioned new ontology overlaps with several existing ontologies in various aspects and degrees: they may share instances, common or similar class or property names, or indeed axioms. The only 'reuse' construct supported by the $\mathcal{OWL}$ languages is to globally import the whole of an ontology. Of course, this is rather undesirable as it is too coarse, i.e. it (often) imports too much.

The general subject of our work are two problems of ontology reuse: firstly, the **matching process**, i.e. how to identify the overlapping fragments of ontologies, and secondly, the **merging process**, i.e. how to merge the overlaps of interest into a new ontology.

Both problems are central issues in the research area of ontology matching [5]. Our contribution regarding the first problem is a graphical user interface to support the user in identifying knowledge overlaps between multiple ontologies interactively. Regarding the second problem, we present a method to extract these identified overlaps, merge them into a new ontology, and analyse their consistency.

The focus of this paper is on the matching process, whereas the second aspect, the merging process, will only briefly be sketched from the user interaction perspective. Theoretical details as well as implementation descriptions about the extraction and merge process are discussed in our related paper [12]. By "the system" we refer in the following to a software component (at the time of writing an experimental system under constant development) that controls the behaviour of a graphical user interface.

As said above, reusing existing ontologies to create new ontologies is a major goal. Creating a new ontology typically starts with a set of relevant terms (words) denoting ontological entities (classes, instances, properties, etc.) of our interest. The intellectually more demanding effort lies in specifying the right constraints for these entities, i.e., defining a concept hierarchy, domain and range for properties, stating axioms, etc. Reusing this kind of knowledge is therefore the main goal. So the basic idea is a systems that takes as initial input only words and returns all relevant knowledge it can find from the ontology repository. This relevant knowledge should comprise all axioms (specified in other ontologies) formally determining the meaning of these words. Finding the right reusable parts is an interactive process, though, only initialized by these input words that serve as seeds for knowledge aggregation. Very often, however, identical concepts are expressed in different words in different ontologies. Therefore, we have to take synonymy into account. Matching class and property names across ontologies is thus our initial step towards ontology reuse. A survey of ontology matching techniques and system evaluations can be found in [5].[3]

For our purposes, we take advantage of existing systems, in particular FALCON [6], to calculate name correspondences between pairs of ontologies. A correspondence is the outcome of a cross ontology matching process and relates two (ontological entity) names from two different ontologies (with a certain confidence level). The most common relations for such correspondences are equivalence, subclass, and disjointness [5].[4] However, at the current state of our work we are contented with the equivalence relation as the sole correspondence considered; i.e. we ask our ontology matcher (FALCON) to search for name correspondences that can be considered as **synonyms**.

Before the user starts interacting with the system, the repository is preprocessed by a matcher system (FALCON in our case): each ontology is matched against each other in the repository. This results in a list of synonyms for each

[3] Ontology matching and alignment based on statistical methods is a relatively developed field, with yearly competitions since 2004 comparing the various strengths and weaknesses of existing algorithms. See `http://oaei.ontologymatching.org/2009/`

[4] Note that modular ontology languages such as DDLs and $\mathcal{E}$-connections take a more complex approach towards cross ontology relationships [3,10], but are outside the scope of this paper.

ontology pair. This way the system knows all about synonymy in the repository beforehand and not just when the user asks for it. In practice, it turns out that, in our repository ORATE[5], less than 5% of ontology pairs have non-empty synonym lists, which reflects that in more than 95% of the cases two randomly chosen ontologies (according to the matcher) talk about completely different things.

From the list of synonyms belonging to pairs of ontologies we compute sets of synonyms (**synsets** for short) that belong to sets of ontologies: for instance, if FALCON determines the word `transporter` from the `Transport` ontology as a synonym for the word `carrier` in the `Logistics` ontology as well as the word `vehicle` in the `Traffic` ontology, then {`transporter`, `carrier`, `vehicle`} is a synset for the ontologies {`Transport`, `Logistics`, `Traffic`}. Once all synsets of the repository are calculated, the ontology developer can match her initial words (used as seeds for the envisioned ontology) against these synsets. It should be mentioned that this matching process is based on regular expressions—no ontology matcher is involved here. If for instance one of the initial words is `vehicle`, then the system would find it in the synset {`transporter`, `carrier`, `vehicle`} and return the ontologies {`Transport`, `Logistics`, `Traffic`} as recommended candidates for ontology reuse in the subsequent steps of ontology development.

Eventually, the system will find all ontologies related (via the precompiled synsets) to a given set of initial words. The ontology developer may then further restrict or extend the set of ontologies that should be partially reused later on. A graph based presentation of the remaining ontologies and synsets allows the developer to select those elements that should be included or excluded in the following knowledge reuse process.

2 Ontology-Class-Synset Graph

In general, ontology matching can match various constructs of the ontology language and the outcome of the matching can be various kinds of relations between these constructs attached with a confidence value [9,5]. This paper describes work in progress in an early stage, hence it currently only tries to capture the fundamental features of ontology matching: we are only interested in matchings that involve class names and whose result is the synonymy relation (moreover, we forget about the confidence level). Thus, **ontology matching** (in the following simply called matching) in our context means to find synonyms of class names between two ontologies, i.e., the result of a matching is a set of name pairs (synonyms). For instance, an `aviation` ontology may contain the classes `airport`, `cargo`, and `passenger` whereas a `ship transportation` ontology may contain the classes `port`, `freight`, and `passenger` among other classes. A typical matching result would be the set of pairs

$$\{(\texttt{airport}, \texttt{port}), (\texttt{cargo}, \texttt{freight}), (\texttt{passenger}, \texttt{passenger}), \ldots\}$$

[5] `http://ontologies.informatik.uni-bremen.de/`

We distinguish two phases of the matching process: first, a matching tool computes synonyms, and subsequently the developer manually deletes mismatches and adds synonyms not detected by the matcher. We shall call the first phase the **automated matching phase** and the latter the **manual matching phase** (or also the **matching revision phase**). The former is moreover a preprocessing phase since it is performed before the developer starts searching for ontologies to be reused.

Semantic matching is a non-trivial process, so we can never totally rely on the results obtained from automated matching. Syntactic matching leads to results like (`airport`, `port`), which is at least an arguable identification. Lexical matchings like lookups in Thesauri (e.g. WordNet[6]) tend to yield more convincing results like (`cargo`, `freight`). Due to this limited competence of automated matching systems, an efficient support for matching revision is desirable to get the right synonyms.

Essentially, our approach for manual matching revision is to **visualise** the relevant outcome of the automated matching process as a **graph** and let the developer (visually) *manipulate this graph* until it represents the desired synonyms between class names of different ontologies.

Our central data structure representing a matching of several ontologies is a special kind of graph, called **OCS-graph** or **visual hyperontology graph**.[7] It has (1) two type of nodes: **C-nodes** representing classes and **O-nodes** representing ontologies, and (2) two types of edges: **CC-edges** between C-nodes representing synonymy and **OC-edges** between O-nodes and C-nodes saying that the class is part of the ontology. Note that an OCS-graph may contain several nodes with the same label. In case of O-nodes it can happen that different ontologies are labelled by their respective developers with the same name. More importantly for us, though, duplicates of C-nodes indicate possible **homonymy**: the class name `point` may stand for two entirely different concepts, a geometric object, or a unit of counting in scoring a game. **Revising a matching** in the graph representation thus means deletion of any kind of nodes or edges, or adding new CC-edges (compare [7] for a related approach concerning mapping revision and debugging).

The main challenge in the graph presentation, however, is to present to the user only the relevant fragment of the OCS-graph, because even for a small set of ontologies the graph soon becomes very complex so that the user easily gets lost in all its crossing links and cluttered nodes.

One obvious way of complexity reduction in a graph is hiding and revealing nodes together with their in- and outgoing links. We will not talk much about this feature; the basic idea follows the scheme “hide all nodes satisfying property

[6] `http://wordnet.princeton.edu/`

[7] These graphs are a ‘visual’ variant of the corresponding ‘textual’ hyperontology specifications which codify the formal structuring of the theory represented by such a graph. The relationship between the two is described in more detail in [12]. Hyperontologies as distributed, structured and heterogeneous ontologies have been studied in depth in [11].

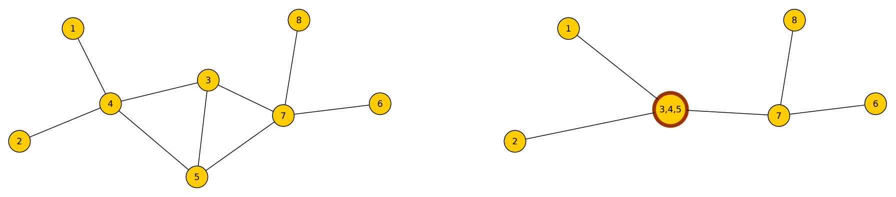

Fig. 1. Collapsed subgraph: the nodes 3, 4, and 5 from the left graph are collapsed to a single node (3,4,5) in the right graph.

P", where P could be, e.g.: "being connected to node C", etc. There is a huge number of possible filter criteria with various areas of applicability (see e.g. [8]), some of which we will make use of in the examples below.

Another common way to reduce complexity efficiently is to collapse connected parts of a graph into single nodes. We consider this technique as one of the most promising for our purpose. Fig. 1 shows schematically how this works in principle.

The only types of subgraphs we want to collapse are syncomponents and C-clones. A **syncomponent** is a maximally CC-edges-connected subgraph that contains only C-nodes. The set of C-nodes extracted from a syncomponent represents a set of synonym class names—we call it a **synset**. A **C-clone** is a maximally connected subgraph of a syncomponent whose C-nodes all have identical labels.[8]

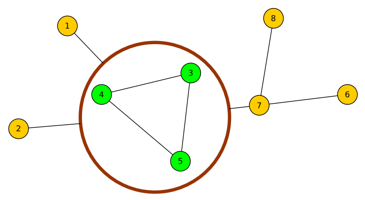

Fig. 2. The nodes 3, 4, and 5 from the left graph of Fig. 1 are collapsed to a graph inside a node.

There are two basic types of visually presenting a graph collapse: reduce to a set of labels (this is depicted in Fig. 1) or to inner graphs (as depicted in Fig. 2). Collapsing to a set of labels reduces more complexity than collapsing to an inner graph: in the latter case, the edges from the outside nodes are reduced whereas in the former case also the edges of the subgraph vanish. Finally, it should be noted that all complexity reduction operations (i.e. *hiding* and *collapsing*) can be performed without any loss of information, i.e., there always exist the corresponding inverse operations of *reveal* and *expand*.

[8] Note that there can be several C-clones within one syncomponent.

3 Graphical Interaction on the Hyperontology Graph

So far we have described in general how we represent a multi ontology matching by a graph and how such a matching can be manually analysed and modified by a set of graph manipulation operations. In this section, we want to illustrate this process by an example.

Let us assume our developer wants to develop an ontology for transportation by reusing knowledge from a repository that contains, among other, several presumably related ontologies like `Logistics`, `Traffic`, `Public transport`, etc. In order to retrieve this knowledge, she starts entering the following initial class names for concepts: `passenger`, `freight`, `vehicle`, `line`, and `route`.

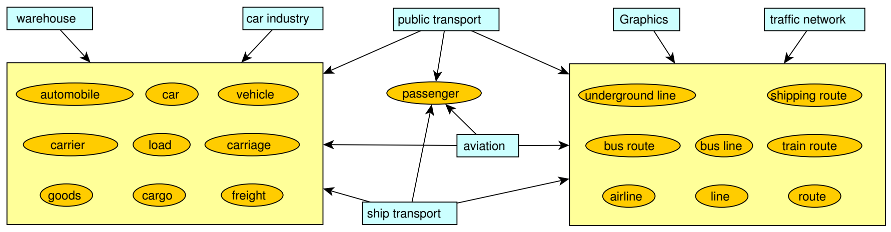

Fig. 3. Presentation of the OCS-graph: blue rectangle = O-node; orange ellipse = C-node. The C-nodes are collapsed to sets of labels that form synsets (big yellow rectangles). Note that `passenger` is a singleton synset.

The system displays the graph as shown in Fig. 3 where all syncomponents are collapsed to sets of labels. Here, and in all following figures, the ellipses represent C-nodes (e.g. `freight`, `passenger`, `line`, etc.). Those ellipses grouped in a big yellow rectangle represent a synset (e.g. `freight` and `vehicle`)[9] whereas a small blue rectangle with a single name in it represents an ontology. There are only arrows from O-nodes to synset boxes or sole C-nodes, meaning that the corresponding classes are contained in the linked ontologies (e.g. the class `freight` is part of the `warehouse` and the `ship transport` ontology). In order to reduce complexity, only those synset boxes are presented that contain synonyms of the class names initially entered by the developer. In our example, we have the above mentioned five initial class names `passenger`, `vehicle`, `line`, and `route`. These are classified (by simple syntactic matching) into three synsets: the left most contains `freight` and `vehicle` among others, the right most contains `line` and `route` among others, and the third in the middle is one that contains `passenger` as the only class name. The condensed graph shows all relevant synsets for a given list of class names together with their contexts, i.e., the ontologies where they occur.

[9] C-nodes outside of a rectangle do not have synonyms (e.g. `passenger`).

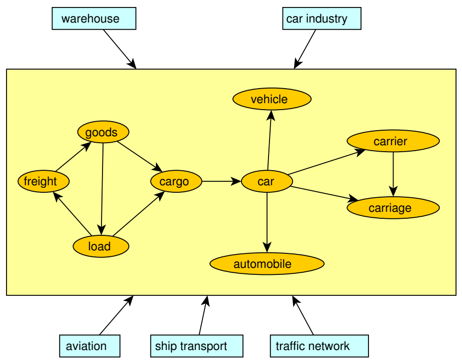

Fig. 4. Graph collapsed to an inner-graph which is a syncomponent.

As mentioned before, the synsets automatically found by a matcher are often unsatisfying. This is also the case for `vehicle` and `freight` in our example: in contrast to the matcher, we would not consider these two words as synonyms for the same concept. To see where this mismatch stems from, we can expand our condensed view of our graph. Fig. 4 shows the expansion of that synset. Arrows between C-nodes in this synset represent the synonymy relation computed by the matcher. Obviously, this relationship is not an equivalence relation, in particular it is not transitive. Otherwise, there should be, for instance, a link from `cargo` to `vehicle`, because these C-nodes are connected via `car`.

Yet, by default, we tacitly assume that the synonymy relation computed by the matcher induces an equivalence relation (i.e., that it's transitive and symmetric closure hold as well). It is the user's obligation, though, to remove (or add) synonymy links from the computed synset to adjust it to commonsense understanding. In our example, an obvious synonymy mismatch is `cargo` to `car` (which typically results from mere partial syntactic matching).

Hence, the user would interactively remove the `cargo`–`car` link and thereby split the synset into two synsets as shown in Fig. 5. Along this separation we see also how the contexts (i.e. the corresponding ontologies) separate, only the `aviation` ontology contains concepts from both synsets. The left-hand synset box of Fig. 5 already comprises a reasonable set of synonyms, whereas the right-hand synset box may suggest further splitting. For this it might be necessary to look more deeply into the (axiomatic) details of the involved ontologies, but this is beyond the scope of this paper.

Let us now turn to the expanded synset depicted in Fig. 6 which contains the class names `line` and `route` (i.e. the right-hand synset box from Fig. 3). This view is more detailed than Fig. 5 as it not just links ontologies to a synset, but moreover to the classes inside the synset. In general, this expansion shows more ontology to class name links, as an ontology itself can already contain synonyms. For instance, in the `public transportation` ontology the matcher has computed `underground line` and `bus line` as synonyms for `line` from

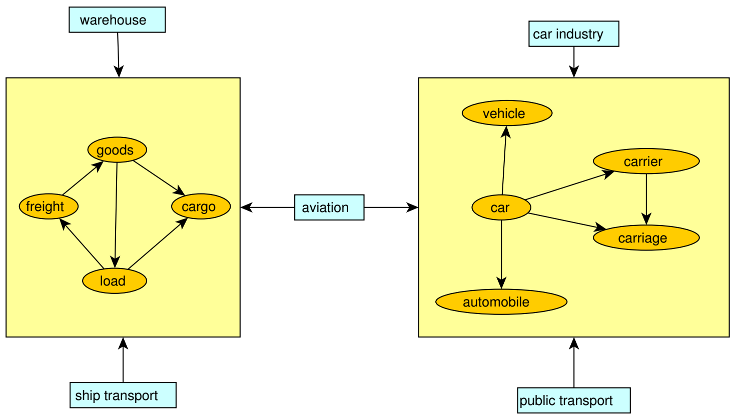

Fig. 5. Collapsed graph with two syncomponent as inner-graphs.

the `ship transportation` ontology. Note again that the matcher has not computed `underground line` and `bus line` directly as synonyms, but due to our interpretation of synonymy as equivalence relation these two class names are indirectly synonyms (due to the transitivity of synonymy). Again, we have several synonymy mismatches. Most of them could be repaired much the same way as described above. But what is new here is the occurrence of homonyms, i.e., one word with two meanings, namely two meanings of the word `line`. Just from the context information we can see that this `line` class name refers to two very different contexts, namely the `graphics` and `ship transportation` ontology.

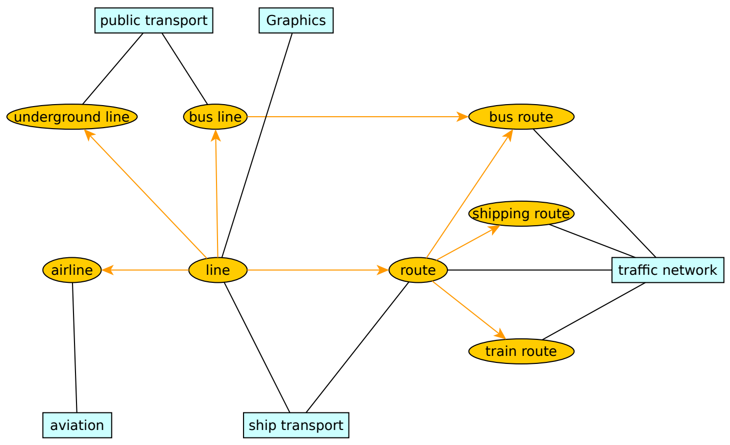

Fig. 6. Graph with two collapsed C-clones: the C-node `line` and the C-node `route`.

Most likely we do not want to integrate the notion of `line` from the graphics ontology into our envisioned transportation ontology. Hence, we want to make explicit the existence of two meanings for a single label. This is done with the further expansion of our diagram Fig. 6 as depicted in Fig. 7, where two nodes with the label `line` are displayed: one linked to the `graphics` and the other to the `ship transportation` ontology. Apart from `line` the class name `route` is also contained in two ontologies, namely `ship transportation` and `traffic network`. In Fig. 6 they are also visible as two C-nodes in Fig. 7.

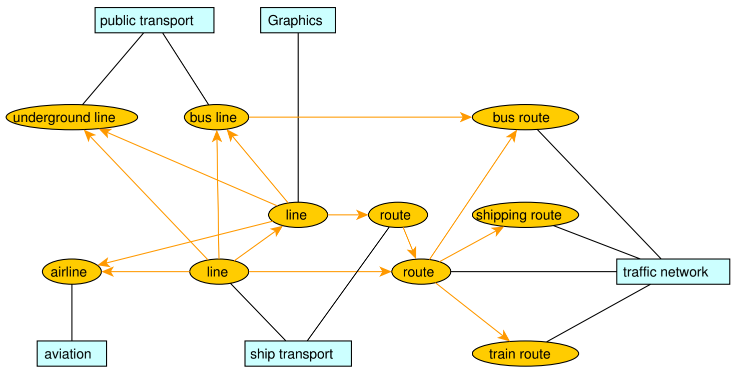

Fig. 7. Fully expanded hyperontology graph.

Comparing Fig. 6 and Fig. 7, we also see that the single synonym link from `line` to `route` expands to two synonym links. In particular, we see that `line` from `ship transportation` has not been linked to `route` from `ship transportation`, but to `route` from `traffic network`—a detail that was not visible in Fig. 6.

Now that we have the fully expanded view of a synset, we can remove all mismatched synonymy links. The final result might look like Fig. 8.

4 Module Extraction and Ontology Synthesis

We said in the previous section that the outcome of the above described procedure is an OCS-graph that serves as a basis for the synthesis of a new ontology that reuses knowledge from the ontology repository. In this section, we sketch the whole process of ontology reuse depicted in Fig. 9 with the graph manipulation being a central part.

Before any interaction with the user takes place, a matcher computes all synonyms between ontologies in the repository and thus builds the initial OCS-graph. Only a part of this graph is presented to the user based on the initially provided class names as described above. Manipulating the OCS-graph is basically the

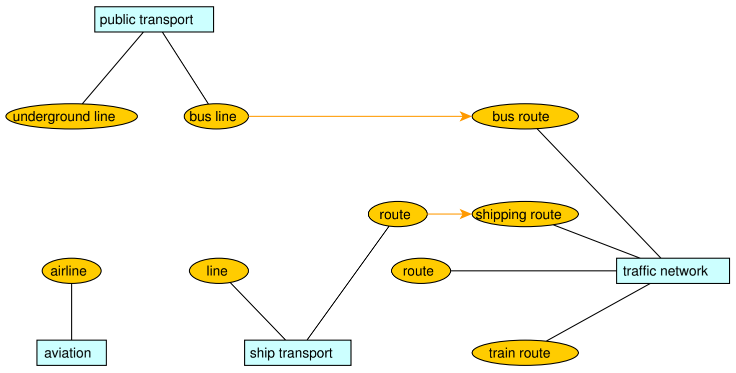

Fig. 8. The final, fully expanded hyperontology graph where all synonymy mismatches are removed.

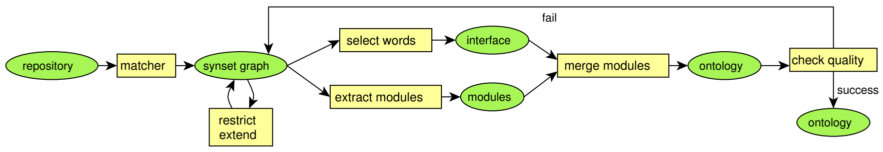

Fig. 9. Flow diagram of the ontology synthesis by reusing knowledge fragments from the ontology repository. Green ellipses represent artefacts that are inputs or outputs to process steps represented as yellow rectangles.

combined action of restriction and expansion that is performed in a cycle until the OCS-graph reflects the intended knowledge core of what should become a new ontology—we call this graph the **seedgraph** for an ontology synthesis. Once the user has extracted the seed graph from the initial OCS-graph, this graph is used to synthesise a new ontology. Each O-node together with all its linked C-nodes in the seed graph are fed into a module extractor (see [15] for a discussion of properties of different kinds of modules). Parallel to this, the user selects from each synset in the seed graph a favourite class name as a representative (since the final ontology should not contain redundant synonyms). The set of all representatives of all synsets constitute the interface for all the modules. From these modules and their interfaces the new ontology is automatically synthesised through so called V-alignments—this process reuses all the knowledge formalised in the modules and uses the interface to remove all redundant synonyms (details can be found in [17,11,12]).

5 Related Work

Ontology matching is the general research field related to our work and [5] is probably the most extensive monograph about this topic. A central online resource to this field is `http://www.ontologymatching.org/`. Both sources list and discuss more than 40 matching tools (not all of them can deal with OWL, though). The main purpose of these tools is to automatically compute alignments—not just regarding synonymy, but also other relations like subsumption and disjointness. At the current stage of our work we have no preference regarding the underlying algorithms of these matchers. We chose Falcon [6] for practical reasons, namely because it was easier to run in batch mode than any other tools we tried. Moreover, it is fast enough to match around 50 ontologies of a repository pairwise against each other within a reasonable amount of time (a few hours on a standard PC). Since our main contribution is the graphical interaction to edit ontology alignments, we want to discuss some of those matching tools that also offer graphical alignment editing. Without making a claim to completeness we list some prominent examples: MapOnto [1], COMA++ [2], Anchor-Prompt [13], and OntoMap [16]. MapOnto is a system for mappings between ontologies and relational or XML schemas. It produces in a semi-automatic way complex mapping formulas expressed in Horn clauses. The user can then choose interactively from alternative mapping formulas. COMA++ is a schema matching infrastructure that provides an extensible library of matching algorithms. The matching operation is a workflow that can be graphically edited. To evaluated the results of different matching operations, special operations are provided. Anchor-Prompt is an extension of Prompt and plugin for Protégé for ontology merging and alignment. OntoMap is a plug-in for the ontology-management platform NeOn Toolkit[10] which supports the creation and management of ontology mappings via a graphical interface. Apart from matching tools, there are ontology repositories that store mappings: CupBoard is a repository that allows users to populate their ontology spaces not only with ontologies, but also with alignments (mappings). Using the Alignment Format [4], alignments can be uploaded for a given (pair of) ontology(ies). An alignment server then offers to CupBoard the necessary features to support the management, evaluation, and even production of alignments. In BioPortal [14], the user can edit, store and retrieve mappings between classes in related ontologies.

Common to all these tools is that always only two ontologies (or schemas) are presented at a time. Hence, semantic overlap can be studied only between two ontologies presented side by side. This is sufficient if the user already knows what pair of ontologies to investigate for potential overlap. Our approach, in contrast, supports the task to first figure out which ontologies have the relevant semantic overlap to a given set of class names. With current systems, this task is very tedious to achieve: one needs to search for ontologies containing the given class names and then scan and analyse all their matchings with other ontologies in the repository step by step. We believe that, with our approach, the user will find much faster the semantic overlaps between ontologies from a repository for a given set of class names because of the simultaneous presentation of matchings involving several ontologies.

[10] `http://neon-toolkit.org`

6 Future Work

We have introduced the notion of a hyperontology graph as a representation for concept matching across ontologies; we demonstrated several task specific techniques of information hiding to cope with the complexity of that graph, and we determined graph modification operations to correct "wrong" concept matchings. The resulting graph can be used to automatically synthesise a new ontology from the explored and user-adjusted fragment of the hyperontology graph.

Synonymy of class names was the only type of link that we considered so far in the hyperontology graph. All our definitions extend naturally to synonymy of property names. However, generalising our approach to other types of matching than synonymy requires more conceptual work. The next matching types we are going to investigate within our approach will be subsumption and disjointness. Adding new types of links to the hyperontology graph increases its complexity and thus calls for new kinds of information hiding techniques.

At the time of writing the system is in its early alpha stage and serves mostly as a proof of concept. It is not yet mature enough for a systematic evaluation w.r.t. usability and scalability, but will be made freely available and accessible as a web-service at a later stage.

Acknowledgements

Work on this paper has been supported by the DFG-funded collaborative research centre SFB/TR 8 'Spatial Cognition' and by the EU-funded OASIS project.

References

1. An, Y., Borgida, A., and Mylopoulos, J. Discovering the semantics of relational tables through mappings. In *LNCS 4244 - Journal on Data Semantics VII* (2006), pp. 1–32.
2. Aumüller, D., Do, H.-H., Massmann, S., and Rahm, E. Schema and ontology matching with COMA++. In *Proc. 24th International Conference on Management of Data (SIGMOD), Software Demonstration* (Baltimore (MD US), 2005), pp. 906–908.
3. Borgida, A., and Serafini, L. Distributed Description Logics: Assimilating Information from Peer Sources. *Journal of Data Semantics 1* (2003), 153–184.
4. Euzenat, J. An API for ontology alignment. In *Proc. 3rd International Semantic Web Conference (ISWC)* (Hiroshima (JP), 2004), vol. 3298 of *Lecture notes in computer science*, pp. 698–712.
5. Euzenat, J., and Shvaiko, P. *Ontology matching.* Springer-Verlag, 2007.
6. Jian, N., Hu, W., Cheng, G., and Qu, Y. Falcon-AO: Aligning ontologies with Falcon. In *Proc. K-CAP Workshop on Integrating Ontologies* (Banff (CA), 2005), pp. 87–93.

7. Jimenez Ruiz, E., Cuenca Grau, B., Horrocks, I., and Berlanga, R. Ontology Integration Using Mappings: Towards Getting the Right Logical Consequences. In *Proc. of the 6th European Semantic Web Conference (ESWC 2009)* (2009), LNCS, Springer.
8. Jünger, M., and Mutzel, P., Eds. *Graph Drawing Software*. Mathematics and Visualization. Springer-Verlag, 2004.
9. Kalfoglou, Y., and Schorlemmer, M. Ontology mapping: the state of the art. *The Knowledge Engineering Review 18*, 1 (2003), 1–31.
10. Kutz, O., Lutz, C., Wolter, F., and Zakharyaschev, M. $\mathcal{E}$-Connections of Abstract Description Systems. *Artificial Intelligence 156*, 1 (2004), 1–73.
11. Kutz, O., Mossakowski, T., and Lücke, D. Carnap, Goguen, and the Hyperontologies—Logical Pluralism and Heterogeneous Structuring in Ontology Design. *Logica Universalis 4*, 2 (2010). Special Issue on 'Is Logic Universal?'.
12. Kutz, O., Normann, I., Mossakowski, T., and Walther, D. Chinese Whispers and Iterative Alignments. In *Proc. of the Fifth International Workshop on Ontology Matching (OM-2010)* (ISWC-2010, November 7, 2010, Shanghai, China), CEUR-WS.
13. Noy, N. F., and Musen, M. A. Anchor-prompt: Using non-local context for semantic matching. In *Proc. of the workshop on ontologies and information sharing at IJCAI-01* (2001), pp. 63–70.
14. Noy, N. F., Shah, N. H., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Montegut, M. J., Rubin, D. L., Youn, C., and Musen, M. A. Bioportal: A web repository for biomedical ontologies and data resources [demonstration], 2007.
15. Sattler, U., Schneider, T., and Zakharyaschev, M. Which kind of module should I extract? In *Proc. of DL-09* (2009), vol. 477, CEUR-WS.
16. Weiten, M. OntoSTUDIO as a Ontology Engineering Environment. In *Semantic Knowledge Management*, J. Davies, M. Grobelnik, and D. Mladenic, Eds. 2009, pp. 50–60.
17. Zimmermann, A., Krötzsch, M., Euzenat, J., and Hitzler, P. Formalizing Ontology Alignment and its Operations with Category Theory. In *Proc. of FOIS-06* (2006), pp. 277–288.