Integration of semantically annotated data by the KnoFuss architecture

Andriy Nikolov, Victoria Uren, Enrico Motta and Anne de Roeck

Knowledge Media Institute, The Open University, Milton Keynes, UK
{a.nikolov, v.s.uren, e.motta, a.deroeck}@open.ac.uk

Abstract. Most of the existing work on information integration in the Semantic Web concentrates on resolving schema-level problems. Specific issues of data-level integration (instance coreferencing, conflict resolution, handling uncertainty) are usually tackled by applying the same techniques as for ontology schema matching or by reusing the solutions produced in the database domain. However, data structured according to OWL ontologies has its specific features: e.g., the classes are organized into a hierarchy, the properties are inherited, data constraints differ from those defined by database schema. This paper describes how these features are exploited in our architecture KnoFuss, designed to support data-level integration of semantic annotations.

1 Introduction

Information integration (also known as knowledge fusion) is a well-recognized problem, initially studied in the database research domain. The challenges here included overcoming both schema-level and data-level heterogeneity [1]. Now, with the emergence of ontologies and the Semantic Web, these problems are studied in the context of semantic data structured according to ontologies. Enterprise-level knowledge management is one of the relevant use cases. In this scenario a corporate ontology is populated both automatically and manually using information from different sources and documents. Thus, while there is no schema-level heterogeneity, the data-level issues are crucial. In this paper we focus on coreferencing: recognizing the cases when different sources refer to the same real-world entities and unifying the URIs of such instances. The data, which has to be integrated, is often noisy: instances have different naming formats, automatic extraction algorithms do not have 100% quality, human editors make occasional mistakes. The output of the coreferencing methods is also not fully reliable. Multiple algorithms have been developed to deal with these problems (e.g., string similarity and set similarity metrics to handle coreferencing) and many existing systems use them in combination (e.g., [2])
While information integration is well-studied in the literature, in the Semantic Web community the research has primarily focused on schema-level matching [4]. Data-level problems have been mostly considered auxiliary and usually tackled together with schema-level integration. The approaches developed in the

database community, on the other hand, do not take into account specific features of semantic data such as the hierarchy of classes. In order to integrate ontological data these features should be taken into account.
In this paper we present our architecture called KnoFuss, which focuses on instance-level integration of data structured according to OWL ontologies. The rest of the paper is organized as follows: in the section 2 we briefly discuss the most relevant existing approaches. Section 3 describes the main principles of the architecture. Section 4 describes the experiments we performed using datasets from the domain of scientific publication. Finally, section 5 summarizes our contribution and outlines directions for future work.

2 Related Work

Information integration has been an important topic of research for a long time, in particular in the database research community. Performing integration requires resolving schema-level and data-level heterogeneity issues [1] where the latter primarily include (i) instance coreferencing, (ii) conflict detection and (iii) inconsistency resolution. There are a number of approaches dealing with these issues. A theoretical basis for solving the instance coreferencing problem (referred as *record linkage*) was defined in [5]. Since then, many solutions were proposed [6], which can be roughly classified into (i) manually constructed rules, (ii) supervised learning algorithms and (iii) unsupervised algorithms. Rule-based approaches are hand-tailored to a specific domain and are hard to reuse. Supervised machine learning algorithms consider record linkage as a specific case of classification [5], [7] or clustering [8] and produce decision models based on a set of training examples. Such algorithms can be applied to different domains but require sufficient training data. Finally, there are unsupervised methods, which include string similarity (Jaro, Levenshtein, Monge-Elkan) and set similarity (cosine, Jaccard, TF-IDF) metrics. Despite being the most generic, however, these algorithms still require their parameters (weights and thresholds) to be configured. In the Semantic Web domain these techniques are included in many ontology matching systems [4].
Since there is no single best algorithm for all domains and the same method has different optimal configuration parameters when applied to different data, one system often employs several methods. The survey of ontology matching systems given in [4] shows that the majority of currently available state-of-the-art ontology matching systems employ a combination of several basic algorithms. The sets of used methods and their configuration information is more often managed internally [3], [9]. However, some frameworks implement a more flexible approach where the library of methods is extensible and configuration parameters can be adjusted. The FOAM framework [10], primarily designed for schema-mapping, includes a special configuration architecture APFEL [11], which learns optimal parameters of matching methods by exploiting user feedback. eTuner [12] aims at the same goal, but constructs an artificially distorted version of the ontology to be mapped. The mappings between the original ontology and its distorted

version (known in advance) are used as a gold standard for the learning algorithm, which produces the configuration parameters of atomic methods. Unlike these two systems, MOMA [2] was specially designed to handle instance-level coreferencing. The system employs an extensible library of matching methods conforming to a uniform interface, invokes them separately and combines their results afterwards. This system, however, assumes a data representation is similar to relational databases without considering semantic links defined by the ontology.

In our view, there is still a space for improvement to adjust existing approaches to the Semantic Web data integration task. As was said, most of the ontology matching systems primarily focus on the schema-level matching and are not optimized for dealing with the data-level issues [4]. In particular, schema-level matching systems, which employ a combination of individual matching techniques, try to select the optimal algorithms' parameters for a pair of ontologies (e.g., [12], [9]). Data-level coreferencing requires more fine-grained tuning: optimal decision models for individuals belonging to different classes of the same ontology might vary. On the other hand, the database record linkage systems do not consider specific properties of ontological data, such as hierarchical relations between classes. Our knowledge fusion architecture KnoFuss has been developed to address these issues.

3 KnoFuss architecture

The current version of the KnoFuss architecture works under the assumption that data to be merged are already structured according to the same OWL ontology. This is a valid assumption, for instance, in the corporate knowledge management scenario mentioned in the Section 1. KnoFuss carries out three main remaining subtasks of the knowledge fusion process:

- **Coreferencing**. The output of this task is a set of mappings between individuals, which are believed to be identical.
- **Conflict detection**. This stage identifies all cases when integration of new data will violate the ontological constraints and diagnoses each inconsistency. The output of the task contains conflict sets: sets of statements contributing to each inconsistency.
- **Inconsistency resolution**. This task actually integrates the data into the target knowledge base, which includes processing inconsistencies.

In this paper we focus on the coreferencing stage. The theoretical basis for our handling of inconsistency resolution is described elsewhere [13]. Each of these subtasks can be performed by different methods, both generic and domain-dependent (e.g., using key attributes or machine-learning models for coreferencing, hand-tailored rules or formal ontology diagnosis for conflict detection). In order to support the selection of optimal methods depending on the domain, the architecture considers the basic techniques for each fusion subtask as

problem-solving methods [14]. Each method represents a separate module, formally described in terms of its inputs, outputs and applicability conditions while its internal behaviour is hidden from the system. The main components of the architecture are:

- *A library of problem-solving methods*, containing algorithms dealing with atomic tasks.
- *A fusion ontology*, describing the information needed to guide the fusion process.

The capabilities of the methods are formally described using the fusion ontology, which is itself represented in OWL and provides two kinds of knowledge structures:

- *Descriptors of problem-solving methods, tasks and application contexts.* This information is used to select and configure methods.
- *Intermediate knowledge structures.* These structures represent meta-level descriptors of the methods' inputs and outputs (e.g., mappings between individuals, sets of conflicting statements).

The fusion process is performed by the system as follows. The system receives as its input a source RDF knowledge base (KB), containing new data to be integrated. Then all tasks of the fusion process (coreferencing, conflict detection and inconsistency resolution) are performed in sequence and produce as a result a set of statements to be integrated into the target KB. Execution of each subtask is controlled by a generic workflow, which (i) selects appropriate methods, (ii) invokes the methods and collects their output and (iii) combines the output of methods filtering out redundancies. In more detail this workflow is explained in the following subsections.

3.1 Method selection

The system starts each atomic task with selection of appropriate methods. The method selection is performed in two phases. First, the system pre-selects all methods, which can potentially be applied to a domain. Appropriate methods are selected based on *method descriptor* objects defined by the fusion ontology. An example of such a descriptor describing the Jaro-Winkler string-similarity coreferencing method is presented in Table 1. The system selects all applicable methods by running the SPARQL queries specified as methods' selection criteria. A method descriptor defines the most general conditions, in which the method can be applied, together with the default configuration parameters. In the example case in Table 1 these include just the threshold and the set of relevant attributes, but other algorithms may involve more complex decision models.
After the set of applicable methods is selected, the system tries to pick up the optimal parameters of the algorithm given the data to which it is applied. It is often the case that the same method can be applied to a wide range of data instances: for instance, string similarity coreferencing algorithms are applicable

Table 1. Coreferencing method descriptor

Method	Label-based Jaro-Winkler matcher
Inputs	*SourceKnowledgeBase* :type *KnowledgeBase*; *TargetKnowledgeBase* :type *KnowledgeBase*;
Outputs	*MergeSets* :type list of *MergeSet* - Set of possible mappings between instances of source and target knowledge bases
Tackles	Coreferencing
Selection criterion	SELECT ?uri WHERE { ?uri rdfs:label ?label }
Reliability	0.9
Description	A generic method, which performs matching based on the label similarity measured using Jaro-Winkler metrics.
Parameters	
Threshold	0.87
Attributes	*rdfs:label*

to any individuals, which have string properties. However, the performance of an algorithm and its optimal settings may differ when it is applied to individuals of a different class. For instance, a Jaro-Winkler string similarity algorithm used to find identical scientific papers must have higher threshold than when it is applied to disambiguate the authors of the papers. Paper titles have generally longer string length and more consistent format, while people's names allow initial abbreviations and titles (like Dr., Prof.), which require the algorithm to be more "tolerant". Thus, as was said in Section 2, a more fine-grained method configuration is needed. The concept of *application context* represents such a configuration. Application contexts specify the parameters of a method when applied to individuals of a particular type. These parameters override the default values defined in the method descriptor. An example of the application context for the Jaro-Winkler string similarity method applied to individuals of class *opus:Publication* describing scientific publications in the SWETO-DBLP ontology is given in Table 2. Application contexts can be organized hierarchically (Fig.

Table 2. Application context example

Method	Label-based Jaro-Winkler matcher
Selection criterion	SELECT ?uri WHERE { ?uri rdf:type opus:Publication . ?uri rdfs:label ?label . }
Linked to class	*opus:Publication*
Reliability	0.95
Parameters	
Threshold	0.93
Attributes	*rdfs:label*, *opus:year*

1) following the taxonomy defined by the domain ontology. Thus, there can be, for instance, a specific configuration for journal articles, which use more specific features then the generic publication coreferencing. By default each application context corresponds to one class in the ontology. Context-dependent method con-

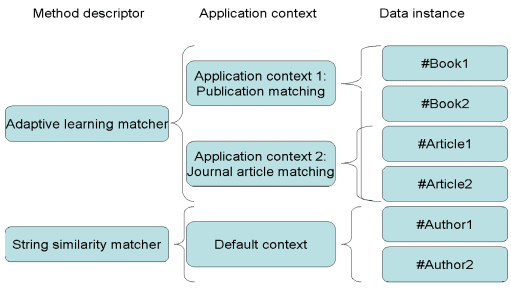

Fig. 1. Method selection using hierarchical application contexts.

figuration requires optimal parameters to be assigned. Assigning them manually is a task requiring significant user effort especially in cases when the domain ontology contains individuals of many different classes. A common way to assign the optimal parameters is to employ machine learning. In the next subsection we discuss how the class hierarchy defined by the domain ontology can be used to enhance the learning procedure.

3.2 Using class hierarchy to learn optimal method parameters

Machine learning can be used to generate optimal parameters for a method applied to a specific class of individuals and estimate its reliability. But in order to train a method to match individuals of a certain class we need sufficient training examples. Obtaining these for each ontological class is often not feasible. Ontological schemata can be exploited in two ways to manage a limited set of training data and assist the learning mechanism:

- Training instances belonging to different subclasses of the same superclass can be combined together.
- Training instances belonging to a subclass can be used to learn a generic decision model for its superclass.

Let's assume that in the ontology we have a class C and its subclasses $C_1 ... C_n$ and for each class we have a set of known individuals D_i. For subsets of these individuals $T_i \sqsubseteq D_i$ we also know the correct identity relations. Pairs of these individuals constitute the training set S_i where pairs of coreferent individuals serve as positive examples and pairs of non-coreferent individuals constitute negative examples. Let f_i represent a set of potentially relevant attributes for each class C_i (e.g., this may include all properties within a range n). Now,

suppose, we only have training instances for a subset of classes $C_1 ... C_n$, i.e., $|S_i| > 0$ where $i \leq m < n$ and $|S_i| = 0$ where $i > m$. The learning algorithm takes as input a set of training examples S and relevant attributes f and produces a decision model $h : (x; y) \rightarrow P(x \equiv y)$. During the configuration phase we train the learning algorithm to produce $m+1$ decision models: for each C_i where $i < m$ and the superclass C. The learning algorithm for the superclass C will take as input the union of all training sets $S = \bigcup_{i=1}^{m} S_i$. The set of relevant features will only contain the features of the class C: $f = \bigcap_{i=1}^{n} f_i$. Then the accuracy of each learned model is evaluated on a set of test examples. The algorithm is included into the library of matching methods and each learned model is described as a separate application context. The reliability of the algorithm in each context is assigned according to the achieved accuracy on the test set. If the accuracy achieved for the model trained for the exact subclass C_i is less than for the superclass C then such a model will not be chosen.

3.3 Method invocation and handling results

After the system has selected applicable methods and has picked the best known configuration parameters for them it proceeds with method invocation. All methods from the selected set are invoked in sequence and their results, structured according to the fusion ontology, are added to the source KB. In cases when the results of methods conflict with each other, only the results produced by the most reliable method are retained. An object representing the result of a method preserves a reference to the method descriptor. Thus, the reliability of the method's output is considered the same as the reliability of the method itself in the context, in which it was invoked. Then the source KB together with the accumulated intermediate information is passed to the next stage.

4 Evaluation

In order to test the system we used the following datasets from the domain of scientific publications:

- AKT EPrints archive[1]. This dataset contains information about papers produced within the AKT research project.
- Rexa dataset[2]. The dataset extracted from the Rexa search server, which was constructed in the University of Massachusets using automatic IE algorithms.
- SWETO DBLP dataset[3]. This is a publicly available dataset listing publications from the computer science domain.

The SWETO-DBLP dataset was originally represented in RDF. The two other datasets were extracted from the HTML sources using specially constructed wrappers and structured according to the SWETO-DBLP ontology (Fig. 3). We performed experiments with the following matching algorithms:

[1] http://eprints.aktors.org/
[2] http://www.rexa.info/
[3] http://lsdis.cs.uga.edu/projects/semdis/swetodblp/august2007/opus_august2007.rdf

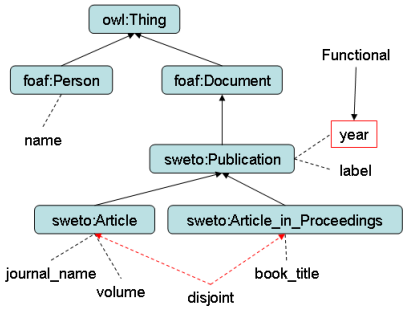

Fig. 2. Class hierarchy in the SWETO-DBLP ontology

- Jaro-Winkler metrics directly applied to the label.
- L2 Jaro-Winkler applied to the label.
- Average L2 Jaro-Winkler over the properties of the class *Publication*.
- Average L2 Jaro-Winkler over all available properties.
- Adaptive learning clustering algorithm employing TF-IDF and N-gram metrics [8].

The Jaro-Winkler algorithm was used as a representative of string matching algorithms (it outperformed Levenshtein on our dataset during the preliminary tests). L2 Jaro-Winkler is a mixture of string similarity and set similarity measures: it tokenizes both compared values, then each pair of tokens is compared using the standard Jaro-Winkler algorithm and the maximal total score is selected. It is able to work in cases when the order of words in a multi-word string value is not important for establishing their identity (e.g., "Enrico Motta" and "Motta, Enrico"). The set of relevant attributes included all immediate datatype property values. We assumed that the algorithms did not have any domain-specific knowledge, so such common techniques as analyzing co-authors to disambiguate a person were not used. The links between papers and authors also were not exploited because of this. First, we trained each algorithm to recognize matching individuals of each immediate class (*Person*, *Article* and *Article_in_Proceedings*). Then, we combined the individuals of classes *Article* and *Article_in_Proceedings* and performed the tests for their superclass *Publication* (only the properties common for the superclass were used in this case). The results of our tests are given in the Table 3. As a performance metric we used the commonly employed F1 measure, which combines precision and recall. Standard deviation of this measure obtained after 5 tests (σ) is given to indicate the robustness of the algorithm in each case. The results indicate that a decision model over the combined dataset (*Publication*) was usually more robust considering the standard deviation in comparison with classes with few training instances (*Article*), although the performance was sometimes lower due to spurious mappings between instances belonging to different classes (for the clustering method). The later problem, however, is handled at the inconsistency resolution stage.

However, there are also important factors, which limit the possibility for the

Table 3. Test results: coreferencing.

Datasets	Article		Article_in_Proceedings		Publication		Person	
	F1	σ	F1	σ	F1	σ	F1	σ
AKT/Rexa								
Direct Jaro-Winkler (label)	0.92	0.09	0.85	0.03	0.87	0.01	0.29	0.01
L2 Jaro-Winkler (label)	0.89	0.07	0.9	0.01	0.9	0.01	0.84	0.01
L2 Jaro-Winkler (label+year)	0.9	0.06	0.92	0.02	0.93	0.03		
L2 Jaro-Winkler (all)	0.48	0.11	0.74	0.03				
Clustering	0.69	0.39	0.85	0.043	0.82	0.045		
AKT/DBLP								
Direct Jaro-Winkler (label)	0.87	0.05	0.94	0.01	0.93	0.02	0.10	0.004
L2 Jaro-Winkler (label)	0.66	0.1	0.52	0.03	0.55	0.01	0.63	0.03
L2 Jaro-Winkler (label+year)	0.88	0.07	0.88	0.01	0.89	0.02		
L2 Jaro-Winkler (all)	0.24	0.06	0.54	0.03				
Clustering	0.75	0.16	0.9	0.03	0.83	0.09		
Rexa/DBLP								
Direct Jaro-Winkler (label)	0.9	0.04	0.91	0.01	0.92	0.01	0.90	0.003
L2 Jaro-Winkler (label)	0.7	0.03	0.7	0.01	0.7	0.01	0.72	0.004
L2 Jaro-Winkler (label+year)	0.93	0.02	0.88	0.05	0.89	0.01		
L2 Jaro-Winkler (all)	0.89	0.02	0.89	0.02				
Clustering	0.86	0.04	0.89	0.01	0.89	0.03		

methods' reuse. First, in order to reuse coreferencing methods between classes linked into a hierarchy we have to assume that the properties significant for identifying the objects are inherited. While this is a common pattern, which holds in our scenario, it may not be the case in some ontologies. Second, differences between property values' formats in different datasets can limit the reuse of the method. For instance, a pair of labels "Sleeman, Derek" in EPrints and "Derek Sleeman" in DBLP could not be caught by the direct Jaro-Winkler algorithm, which was the best for the pair Rexa/DBLP.

5 Conclusion and future work

In this paper we have presented the architecture KnoFuss aimed at performing data-level fusion of OWL knowledge bases. We consider the method selection mechanism, which allows the system to pick and configure individual methods considering the class hierarchy, as the main contribution described in the paper. We have implemented the architecture described in the paper and performed initial tests using datasets from the publications domain.

There are several directions we consider important for future work. First, the current version of the system operates under the assumption that the knowledge bases to be integrated are structured according to the same ontology. In order to be used in a multi-ontology environment the architecture must be able to incorporate the results produced by ontology matching algorithms. Another issue concerns the coreferencing method selection, which at the moment is primarily

based on the class to which an individual belongs. However, it is often the case that method performance may significantly depend on the data format, which in turn depends on the data source. In order to improve the system's performance, an efficient mechanism to pick up an optimal method for an unknown source has to be implemented.

6 Acknowledgements

This work was funded by the X-Media project (www.x-media-project.org) sponsored by the European Commission as part of the Information Society Technologies (IST) programme under EC grant number IST-FP6-026978. The authors would like to thank Steffen Rendle and Karen Tso for providing their object identification tool [8].

References

1. Kim, W., Seo, J.: Classifying schematic and data heterogeneity in multidatabase systems. IEEE Computer **24**(12) (1991) 12–18
2. Thor, A., Rahm, E.: MOMA - a mapping-based object matching system. In: 3rd Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA (2007)
3. Straccia, U., Troncy, R.: oMAP: Combining classifiers for aligning automatically OWL ontologies. In: 6th International Conference on Web Information Systems Engineering (WISE). (2005)
4. Euzenat, J., Shvaiko, P.: Ontology matching. Springer-Verlag, Heidelberg (2007)
5. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. Journal of American Statistical Association **64**(328) (1969) 1183–1210
6. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering **19**(1) (2007) 1–16
7. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003), Washington DC (2003) 39–48
8. Rendle, S., Schmidt-Thieme, L.: Object identification with constraints. In: 6th IEEE International Conference on Data Mining (ICDM). (2006)
9. Jian, N., Hu, W., Cheng, G., Qu, Y.: Falcon-AO: Aligning ontologies with Falcon. In: K-CAP Workshop on Integrating Ontologies, Banff (CA) (2005) 87–93
10. Ehrig, M.: Ontology Alignment: Bridging the Semantic Gap. Springer, New York, NY US (2007)
11. Ehrig, M., Staab, S., Sure, Y.: Bootstrapping ontology alignment methods with APFEL. In: 4th International Semantic Web Conference (ISWC-2005). (2005)
12. Lee, Y., Sayyadian, M., Doan, A., Rosenthal, A.S.: eTuner: Tuning schema matching software using synthetic scenarios. VLDB Journal **16** (2007) 97–122
13. Nikolov, A., Uren, V., Motta, E., de Roeck, A.: Using the Dempster-Shafer theory of evidence to resolve ABox inconsistencies. In: Workshop on Uncertainty Reasoning for the Semantic Web, ISWC 2007, Busan, Korea (2007)
14. Motta, E.: Reusable Components for Knowledge Modelling. Volume 53 of Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam (1999)