Ontological Re-Classifications of Instances

Radek Mařík

CA CZ, s.r.o., CA Labs, V Parku 2316/2
148 00 Praha, Czech Republic
Radek.Marik@ca.com

Abstract. A fast and easy way of ontology introduction into knowledge management systems of large enterprises is highly desirable as companies need to keep pace with rapid organization, business, and technology changes. Nevertheless, ontology building, related instance classifications and tagging are tedious ongoing processes that are tackled with a number of approaches. A simple semi-automated technique for transferring instances from a given source ontology to another one based on instance classification using a partial bridge ontology is proposed in this paper. We demonstrate the method using widely available tool Protégé integrated with Pellet reasoner without requiring of an additional complex software implementation.

1 Introduction

Accessing, browsing, searching, and understanding information pools in large corporations might become very difficult if an appropriate knowledge management system is not implemented. Information sources in enterprises are often kept in a number of heterogeneous databases and documents accessible through dedicated applications or Web-enabled interfaces including different content management systems. It is not easy, if not impossible, to impose a uniform access system with a unique terminology [6][16]. A real situation might be often worse because of a number of acquisitions made by large enterprises.

There are a number of possible technologies that create appropriate environments. A majority of such utilities and tools are founded around the notion of ontology, i.e. an explicit shared specification of conceptualization in a given domain, whereby a conceptualization is a set of objects and entities existing in the given domains and that are characterized through properties and relationships and other constraints [5].

However, an explicit form of ontologies of a number of domains has not been created or they cover domains only partially. This leads naturally to ontology development that is not easy and implies a not negligible cost. A number of methods have been published. We can clearly recognized two main streams. The first group represents a number of articles, papers, books and WWW pages recommending a suitable sequence of steps how a team can create efficiently an ontology in manual way [10]. The second stream deals with semi-automated or

© Václav Snášel (Ed.): Znalosti 2008, pp. 136–147, ISBN 978-80-227-2827-0.
FIIT STU Bratislava, Ústav informatiky a softvérového inžinierstva, 2008.

fully automated ontology learning and discovery [1][8][7][11]. Typically, methods are focused on a creation of a semantic bridge that relates several ontologies where the bridge is constructed by matching ontologies only without a support of instance data [17][14]. Most of methods compare the similarity of source ontology structures or similarities between concepts of the source ontologies and their data instances [9][4]. Examples of *source based* mapping tools are PROMPT, Chimaera, and ONION. FCA-MERGE [18] utilizing concept lattices or GLUE [3] using distributions of the concepts in data instances are examples of *instance based* mapping tools.

Instance description based on a given ontology seems to be a nightmare for many businessmen as nobody wishes to add additional burden on their customers asking them to tag any piece of information according to a given ontology. However, such an approach seems to be viable in the WWW world. Nevertheless, a lot of data in the industrial world is stored as structures or semi-structures with defined properties. Processes manipulating with data can often produce a description based on a very weak ontology in such cases.

A step of ontology mapping determination belongs to traditional arms how to convert instance description into another ontology. However, searching for ontology mapping belongs to very difficult tasks.

In this paper we present a simple but powerful method that allows converting data expressed in a very weak ontology into instances with a rich ontology. The method is based on a re-classification of instances with both source and target ontologies merged through a partial bridge ontology. In step by step manner weak ontology concepts are mapped into the target ontology. The approach is presented for class generalization and class specialization. A rule based extension might be used for identification of relations (not discussed in this paper). We demonstrate in the experimental part using a real data set that about 70 pairs of source and target concepts allow to map a majority of over 1000 source weak concepts into 12 target concepts.

The paper is organized as follows. Section 2 introduces a model example dealing with company organization. We propose a novel approach in Section 3. Experimental achievements are presented in Section 4. Section 5 concludes the paper.

2 Exemplar model

We will use a very simple example to demonstrate our approach. Let us assume we have source data dealing with a large corporation organization. We will focus only on specific aspects of the organization to simplify the case.

Let us define our example in the following way. Each employee belongs to a department. Each employee is also assigned with a job position. Taxonomies of departments and employees are different in the source and target ontologies. Our goal is to map the source structure into the target ontology that determines also a structure of departments regarding their types as the source ontology defines only a general type of department. We also want to unify job position titles.

As we are going to present the approach using the Protégé OWL plugin [13] combined with Pellet reasoner [12][15], we simplify our example even further by using a common ontology (further referenced with prefix prop:) that defines shared properties between the source and target ontologies. owl:samePropertyAs might be used in practice for creating a property bridge in a similar way as we will suggest further for classes. Thus, the shared ontology is very simple containing just two properties belongsToDepartment and hasEmployee:

$$\mathsf{hasEmployee} \equiv \mathsf{belongsToDepartment}^{-1}$$

2.1 Source Ontology and Raw Data

Raw data does not provide any additional information on department than its identification. For example, real department names might be too cryptic to derive any additional properties. Actual data for large enterprises with a number of employees over 30.000 might contain over thousands of such department names and thousands of different job position titles. Similar cases can be found in other domains describing, for example, products or software service APIs.

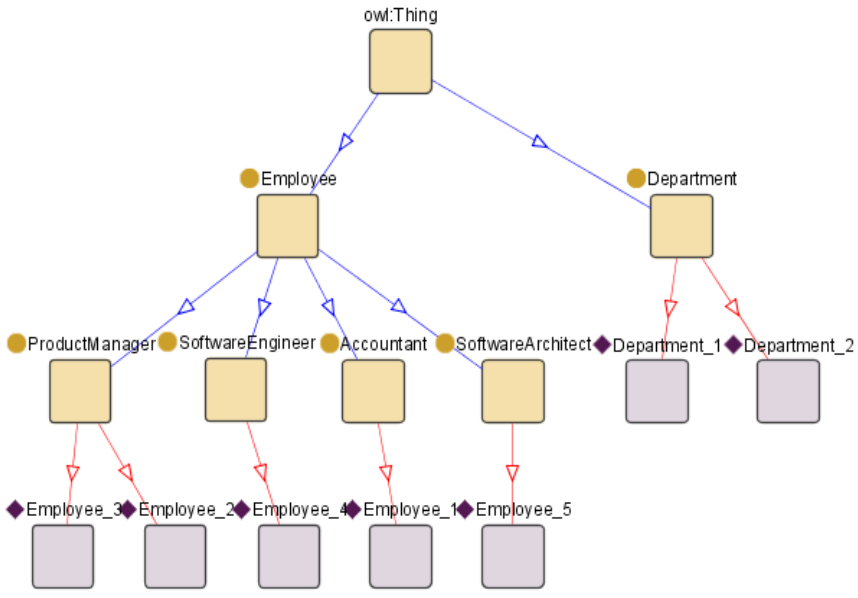

Fig. 1. The source ontology contains two general concepts, Department and Employee. Nothing more is known about concept Department. However, concept Employee has a very rich set of specilized concepts: ProductManager, Accountant, SoftwareArchitect, and SoftwareEngineer. The leaf rectangles tagged with diamonds represent asserted instances of given concepts

The source ontology (further with prefix RAW:) might be defined using DL syntax in the following way:

$$\begin{aligned}
\textsc{Department} &\sqsubseteq \exists \mathsf{prop{:}hasEmployee}.\textsc{Employee}\\
\textsc{Employee} &\sqsubseteq \exists \mathsf{prop{:}belongsToDepartment}.\textsc{Department}\\
\textsc{SoftwareArchitect} &\sqsubseteq \textsc{Employee}\\
\textsc{ProductManager} &\sqsubseteq \textsc{Employee}\\
\textsc{Accountant} &\sqsubseteq \textsc{Employee}\\
\textsc{SoftwareEngineer} &\sqsubseteq \textsc{Employee}
\end{aligned}$$

We will also need some instances of employees and departments as defined in the following table to demonstrate the method.

$\textsc{Accountant}(\textit{Employee\_1})$
$\textsc{ProductManager}(\textit{Employee\_2})$
$\textsc{ProductManager}(\textit{Employee\_3})$
$\textsc{SoftwareEngineer}(\textit{Employee\_4})$
$\textsc{SoftwareArchitect}(\textit{Employee\_5})$
$\textsc{Department}(\textit{Department\_1})$
$\textsc{Department}(\textit{Department\_2})$
$\mathsf{prop{:}belongsToDepartment}(\textit{Employee\_1}, \textit{Department\_1})$
$\mathsf{prop{:}belongsToDepartment}(\textit{Employee\_2}, \textit{Department\_1})$
$\mathsf{prop{:}belongsToDepartment}(\textit{Employee\_3}, \textit{Department\_2})$
$\mathsf{prop{:}belongsToDepartment}(\textit{Employee\_4}, \textit{Department\_2})$
$\mathsf{prop{:}belongsToDepartment}(\textit{Employee\_5}, \textit{Department\_2})$

The overall structure of source data is captured in Figure 1.

2.2 Target Ontology

The target ontology operates with a different set of job positions allowing to make abstractions and to address subsets of employees. The target ontology defines also types of departments regarding to activities their employees perform. It is sufficient to define just two types of job positions for demonstration purposes, DEVELOPER and BUSINESSMAN. Similarly, we use only two types of departments, SALEDEPARTMENT and DEVELOPMENTDEPARTMENT. Both department types are described as defined classes. Thus, the simplified target ontology (further prefixed with TGT:) is determined by the following axioms:

$$\begin{aligned}
\textsc{Department} &\sqsubseteq \exists \mathsf{prop{:}hasEmployee}.\textsc{Employee}\\
\textsc{SaleDepartment} &\equiv \exists \mathsf{prop{:}hasEmployee}.\textsc{Businessman} \sqcap \textsc{Departmen}\\
\textsc{DevelopmentDepartment} &\equiv \exists \mathsf{prop{:}hasEmployee}.\textsc{Developer} \sqcap \textsc{Department}\\
\textsc{Employee} &\sqsubseteq \exists \mathsf{prop{:}belongsToDepartment}.\textsc{Department}\\
\textsc{Developer} &\sqsubseteq \textsc{Employee}\\
\textsc{Businessman} &\sqsubseteq \textsc{Employee}
\end{aligned}$$

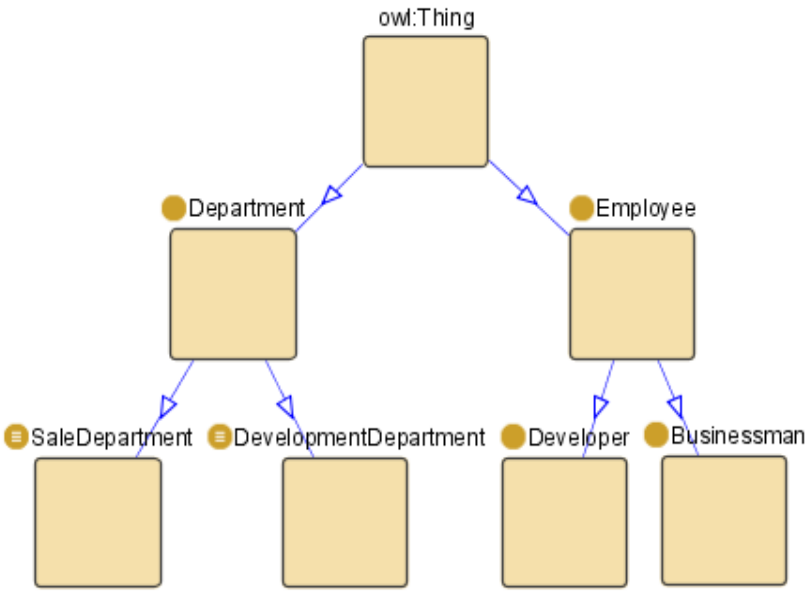

Fig. 2. The target ontology contains two general concepts, DEPARTMENT and EMPLOYEE. Both concepts have two specializations, i.e. DEVELOPMENTDEPARTMENT and SALEDEPARTMENT for DEPARTMENT, BUSINESSMAN and DEVELOPER for EMPLOYEE

The target ontology graph is depicted in Figure 2.

3 Step-wise Ontology Mapping

Raw source data typically provides classes that are too specific or too general than processes using target ontologies require. Traditional methods manually define a mapping between the source and target ontology in which the knowledge engineer assigns a class from the target ontology to a given class of the source ontology. Although such a process gains a quite reliable mapping, it is very time consuming and costly. There are a number of methods that attempt to find a mapping in automatic way. They might produce spurious assignments that are difficult to detect or locations of inconsistencies cannot be found easily.

In this section we will show that if source data are sufficiently redundant, i.e.

- the source ontology contains significantly more concepts than the target ontology and
- the source ontology is not too different from the target ontology

then we can use a semi-automatic method. The method uses instances classified according source ontologies as links between concepts. We will present the method on class generalization and specialization mapping. The key idea is that

we do not need to map all classes but only few of them. For example, to recognize a type of a department instance according to the target ontology we need to define a map between the source and target job positions. However, we do not need to create a total function covering all source ontology job positions.

It is sufficient to select a subset of classes which instances cover all instances of departments with at least one job position instance. For example, such a process can be accomplished using a set coverage tool or guided by formal concept analysis tools. One also can employ a sub-optimal procedure by selecting a source job position in one by one manner while creating a map between the source ontology job position and the target ontology job position and re-classifying department instances. A job position covering the most unclassified departments can be selected as the next one if a map between source and target job position classes is clear.

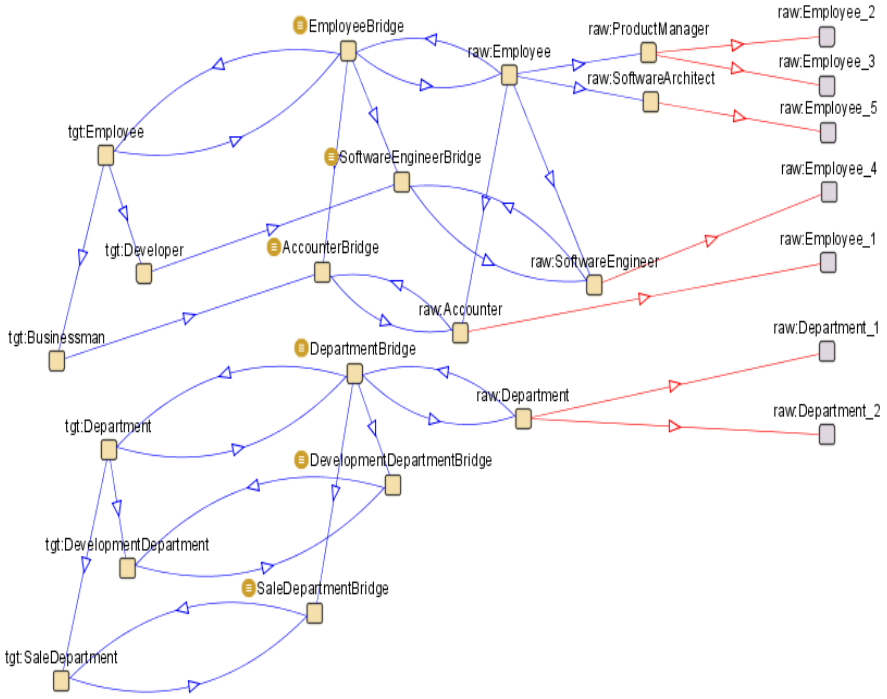

Fig. 3. Concepts of the source (on the left side) and target (the right side) ontologies are related through concepts of the bridge ontology (the two middle trees). The right most rectangles linked the the right source ontology represent available instances.

In our example, although ProductManager cover both department instances, it does not provide an obvious recognition of department type. So, we select Accountant for SaleDepartment and SoftwareEngineer for DevelopmentDepartment. We use a notion of bridge ontology for a description

of mapping between two ontologies. **Bridge ontology** in our approach is an ontology that describes relationships between classes and properties of several (in our case two) ontologies. Equivalence and subsumption belongs among typical relations used to define a bridge ontology. Let us define our example bridge ontology as follows:

$$
\begin{aligned}
\textsc{EmployeeBridge} &\equiv \textsc{raw:Employee}\\
\textsc{EmployeeBridge} &\equiv \textsc{tgt:Employee}\\
\textsc{AccountantBridge} &\sqsubseteq \textsc{EmployeeBridge}\\
\textsc{AccountantBridge} &\sqsubseteq \textsc{tgt:Businessman}\\
\textsc{AccountantBridge} &\equiv \textsc{raw:Accountant}\\
\textsc{SoftwareEngineerBridge} &\sqsubseteq \textsc{EmployeeBridge}\\
\textsc{SoftwareEngineerBridge} &\sqsubseteq \textsc{tgt:Developer}\\
\textsc{SoftwareEngineerBridge} &\equiv \textsc{raw:SoftwareEngineer}\\
\textsc{DepartmentBridge} &\equiv \textsc{raw:Department}\\
\textsc{DepartmentBridge} &\equiv \textsc{tgt:Department}\\
\textsc{SaleDepartmentBridge} &\sqsubseteq \textsc{DepartmentBridge}\\
\textsc{SaleDepartmentBridge} &\equiv \textsc{tgt:SaleDepartment}\\
\textsc{DevelopmentDepartmentBridge} &\sqsubseteq \textsc{DepartmentBridge}\\
\textsc{DevelopmentDepartmentBridge} &\equiv \textsc{tgt:DevelopmentDepartment}
\end{aligned}
$$

The concepts of the source and target ontologies linked through the bridge ontology concepts are depicted in Figure 3.

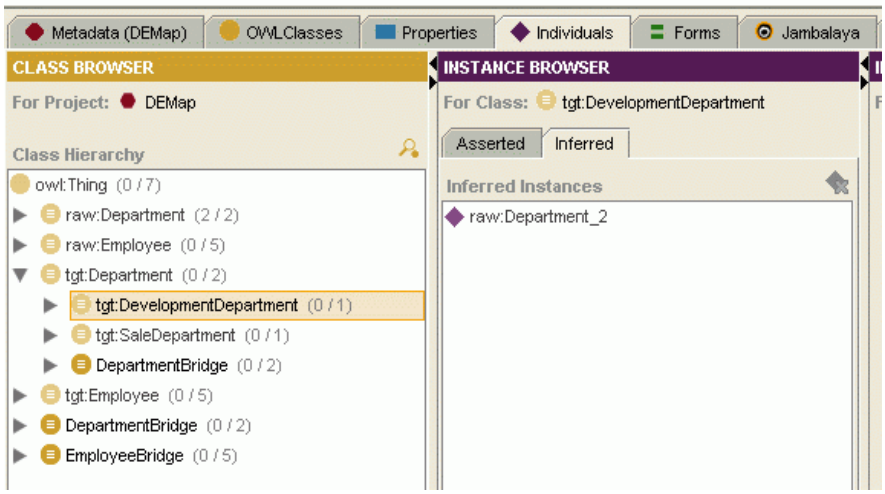

Fig. 4. Concept TGT:DEVELOPMENTDEPARTMENT has no asserted instances (the left CLASS BROWSER pane). However, we can find one department instance (RAW:DEPARTMENT_2) under the `Inferred` pane of the INSTANCE BROWSER on the right side. The corresponding change (0/1) is captured in the parentheses at the concept in the CLASS BROWSER

Now, we can re-classify instances of the source ontology. Protégé combined with Pellet reasoner derives easily that RAW:DEPARTMENT_1 can be classified as TGT:SALEDEPARTMENT and RAW:DEPARTMENT_2 is inferred to be instance of TGT:DEVELOPMENTDEPARTMENT as depicted in Figure 4.

In other words, we infer using contextual information that a type of department can be specialized.

Class generalization is also possible. Let us add necessary condition to the department bridges specifying that all employees of DEVELOPMENTDEPARTMENT must be developers and all employees of SALEDEPARTMENT must be businessmen.

$$\text{SALEDEPARTMENTBRIDGE} \sqsubseteq \forall \text{prop:hasEmployee}.\text{TGT:BUSINESSMAN}$$
$$\text{DEVELOPMENTDEPARTMENTBRIDGE} \sqsubseteq \forall \text{prop:hasEmployee}.\text{TGT:DEVELOPER}$$

If we re-classify instances again then one can find easily out the employees belonging to the DEPARTMENT_1 are developers and the employees from DEPARTMENT_2 are businessmen as seen in Figure 5.

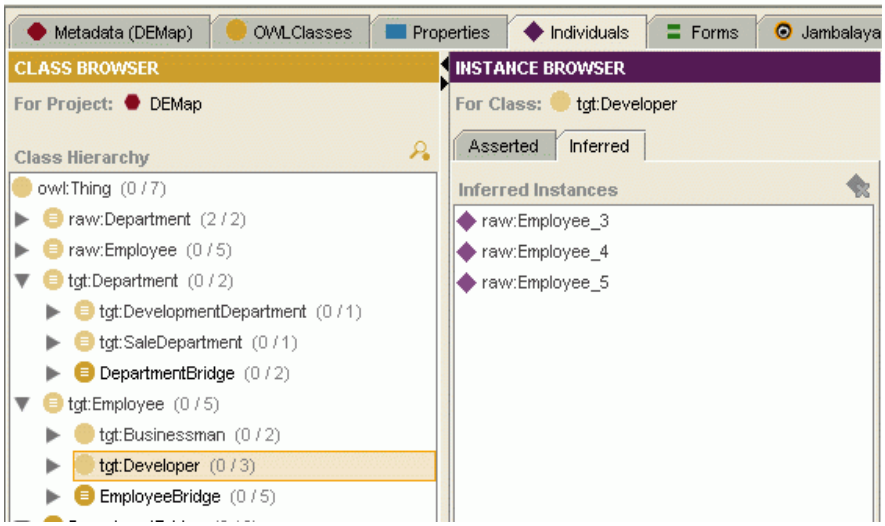

Fig. 5. Concept TGT:DEVELOPER has no asserted instances (the left CLASS BROWSER pane). However, we can find three employees under the `Inferred` pane of the INSTANCE BROWSER on the right side. The corresponding changes are captured in the parentheses at the concepts in the CLASS BROWSER

In this way, assuming the correctness of the bridge, we can infer that source concept of SOFTWAREARCHITECT is related to DEVELOPER concept. So, DEVELOPER concept covers both SOFTWAREENGINEER and SOFTWAREARCHITECT at least and it might be considered as their generalization. This information can

be further utilized and a system helping to create a map between two ontologies can propose that DEVELOPER subsumes SOFTWAREARCHITECT.

Nevertheless, we can also detect a spurious inference. We have two PRODUCTMANAGER employees. One of them is classified as DEVELOPER, the other one is inferred to be BUSINESSMAN. Possible interpretations or contingent modifications of this effect depend heavily on a given case. However, we should stress that if we create such an ontology map than we can detect inconsistency quite easily.

4 Experimental results

The method was applied to data from a real domain with a structure of ontology description similar to the structure of the presented example used in the previous sections. The data set contained 16.456 instances tagged with 1073 different subtype labels of type S1. Those instances were grouped into 1641 sets tagged with 787 different subtype labels of type S2.

We identified only 73 assignments between subtypes of type S1 from the source ontology and 12 more general concepts of type T1 from the target ontology in incremental way. The number of target ontology subtypes of T2 related to the type S2 was also set to 12. This small number of equivalence definitions resulted in classification coverage of 1403 sets and 15.197 instances with regard to the target ontology. Similar achievements might be observed in type translations.

The efficiency of the method is clearly evident from Table 7. Instead of a development of over 2000 equivalences we needed just 73 only (i.e. 4%) to cover over 80% of instance data translations and about 70% of concept translations mapping the source ontology into the target ontology.

Table 7. A proper selection of concept bridges results in high coverages of source concepts and instances by target ones. We had to align only 4% of all equivalences in our case

Entity	Source data count	Classified	Coverage
Instances	16.456	15.197	92%
Sets	1641	1403	85%
Instance types	1073	658	61%
Set types	787	599	76%

The above experiment was conducted using a Python script coding re-classification rules in 11 seconds. In practice, we would like to avoid programming of such scripts. Unfortunately, even current state of art reasoners like Pellet (version 1.5.0) cannot cope with such a volume of data. We have selected slices

of our experimental data to demonstrate processing time increase. Source data volumes and result times for consistency checking (CC Time), taxonomy classification (TC Time), inferred types computation (ITC Time) are shown in Table 8. The processing was performed on a PC with Intel 2.16 GHz, 2 GB RAM, MS Windows XP with Protégé 3.3.1 integrated with Pellet using DIG [2].

Table 8. Instance classification (ITC) times for several data slices taken from the original data set. Consistency checking and taxonomy classification time increase also, but they are negligible in comparison with ITC time

Slice	1	2	3	4	5
Instances	36	172	289	527	759
Sets	4	23	32	50	79
Instance types	18	35	60	122	143
Set types	1	21	23	33	41
CC Time[s]	0.24	0.34	0.48	0.69	0.86
TC Time[s]	0.33	0.45	0.47	0.77	0.92
ITC Time[s]	0.88	29.05	79.16	291.85	719.98

The exponential dependency is obvious from Graph 6 with logarithmic scales on both axes.

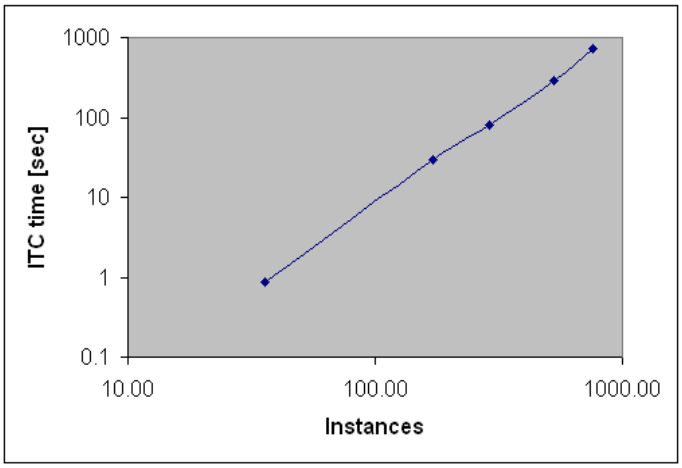

Fig. 6. Instance classification belongs to difficult tasks performed by only few reasoners. Pellet belongs to them. As the number of instances of type S1 grows time for inference grows even faster. Both axes are in logarithmic scales. The observed linear dependency suggests an exponential dependency between number of instances and inference time

5 Conclusions

Transforming descriptions of instances from an ontology to another one is generally a difficult task that is often performed using a mapping between the ontologies. In this paper, we propose a method based on practical concept redundancy in raw instance data that allows specifying only a part of the mapping as bridge ontology. Bridge ontology might describe only a significantly smaller set of relationships between the ontologies. The rest of mapping can be suggested by the system after the re-classification of instances. In such a case, the user can just confirm the proposed relationship and she can save significant effort that must be paid if she would search for all mappings between related concepts.

A practical utilization of the proposed technique depends heavily on reasoner performance during inferring types of individuals. One must assess carefully input data volumes with regard to current implementations of reasoners. We are going to focus also on other entities of OWL DL language in future and to deal with optimization techniques of mapping selections.

Acknowledgements

We would like to thank Zdenek Kouba and Petr Kremen from the CTU FEE, Prague for valuable discussions clarifying some steps of our approach. We also thank our CA Labs colleagues, especially Gabby Silberman for his encouragement and Kirk Wilson for his inputs during this work.

References

1. Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Record*, 35(3):34–41, September 2006.
2. DIG interface. `http://dig.sourceforge.net/`, October 2007.
3. AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. *Handbook on Ontologies in Information Systems*, chapter Ontology Matching: A Machine Learning Approach, pages 385–403. New York. Springer, 2004.
4. Marc Ehrig and Steffen Staab. Qom - quick ontology mapping. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference*, volume 3298 of LNCS, pages 683–697, Hiroshima, Japan, 2004. Springer.
5. Thomas R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In Nicola Guarino and Roberto Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer Academic Publishers, 1993.
6. Dieter E. Jenz. Better alignment of IT with business - the ontology-based approach. `http://www.bpiresearch.com/WP_BPMAlignment.pdf`, October 2003.
7. Yannis Kalfoglou and W. Marco Schorlemmer. Information-flow-based ontology mapping. In Robert Meersman and Zahir Tari, editors, *CoopIS/DOA/ODBASE*, volume 2519 of *Lecture Notes in Computer Science*, pages 1132–1151. Springer, 2002.

8. Yannis Kalfoglou and W. Marco Schorlemmer. Ontology mapping: The state of the art. In Yannis Kalfoglou, W. Marco Schorlemmer, Amit P. Sheth, Steffen Staab, and Michael Uschold, editors, *Semantic Interoperability and Integration*, volume 04391 of *Dagstuhl Seminar Proceedings*. IBFI, Schloss Dagstuhl, Germany, 2005.
9. Laurel C. Y. Kong, C. L. Wang, and F. C. M. Lau. Ontology mapping in pervasive computing environment. In *International Conference on Embedded and Ubiquitous Computing*, pages 1014–1023, Aizu, Japan, August 2004.
10. Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, Stanford, CA, 94305, March 2001.
11. Natalya Fridman Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70, December 2004.
12. Pellet, version 1.5.0. `http://pellet.owldl.com/`, July 2007.
13. Protege-owl editor, version 3.3.1 (build 430). `http://protege.stanford.edu/`, August 2007.
14. Nuno Silva and Joo Rocha. Service-oriented semi-automatic ontology mapping. *International Journal of Engineering Intelligent Systems; 13(4) ; CRL Publishing*, 13(4):253–258, 2005.
15. Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.
16. John P. Stenbit. DoD net-centric data strategy. `http://www.defenselink.mil/cio-nii/docs/Net-Centric-Data-Strategy-2003-05-092.pdf`, May 2003.
17. Larry M. Stephens, Aurovinda K. Gangam, and Michael N. Huhns. Constructing consensus ontologies for the semantic web: A conceptual approach. *World Wide Web Journal, Kluwer Academic Publishers*, 7(4):421–442, December 2004.
18. G. Stumme and A. Maedche. Fca-merge: Bottom-up merging of ontologies. In B. Nebel, editor, *Proc. 17th Intl. Conf. on Artificial Intelligence (IJCAI '01)*, pages 225–230, Seattle, WA, USA, 2001.