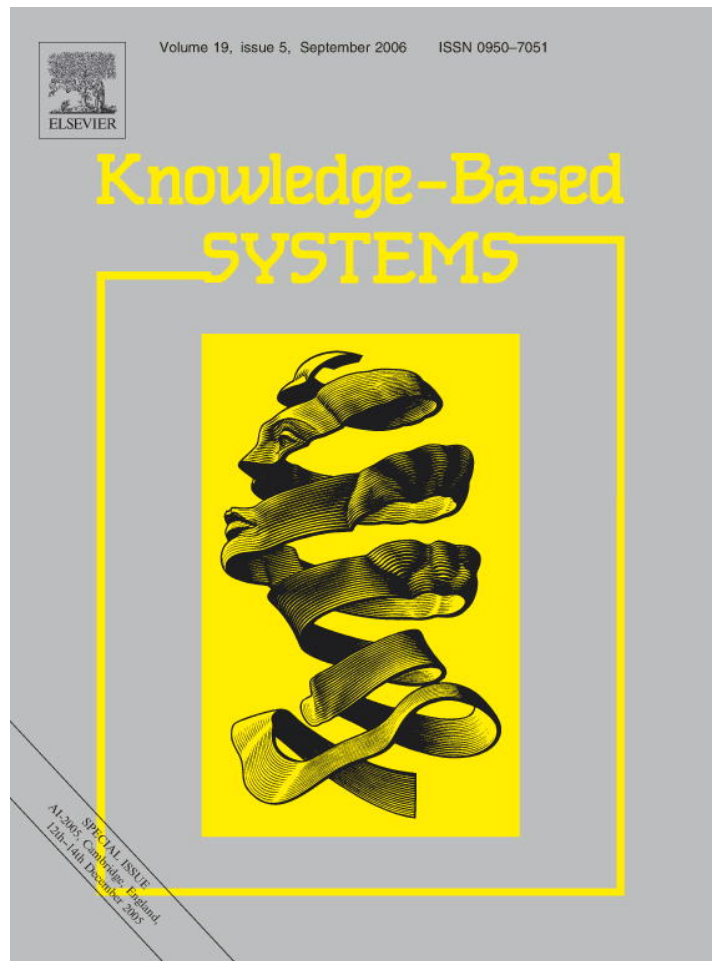


Provided for non-commercial research and educational use only.
Not for reproduction or distribution or commercial use.



This article was originally published in a journal published by Elsevier, and the attached copy is provided by Elsevier for the author's benefit and for the benefit of the author's institution, for non-commercial research and educational use including without limitation use in instruction at your institution, sending it to specific colleagues that you know, and providing a copy to your institution's administrator.

All other uses, reproduction and distribution, including without limitation commercial reprints, selling or licensing copies or access, or posting on open internet sites, your personal or institution's website or repository, are prohibited. For exceptions, permission may be sought for such use through Elsevier's permissions site at:

<http://www.elsevier.com/locate/permissionusematerial>



ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Knowledge-Based Systems 19 (2006) 291–297

Knowledge-Based
SYSTEMS

www.elsevier.com/locate/knosys

Reusing JessTab rules in Protégé

D. Corsar *, D. Sleeman

Department of Computing Science, University of Aberdeen, Aberdeen AB24 3UE, Scotland, UK

Received 28 October 2005; accepted 28 November 2005

Available online 6 February 2006

Abstract

Protégé provides a complete ontology and knowledge base management tool. Along with JESS, JessTab provides one method of rule-based reasoning over a Protégé ontology and knowledge base. However, once JessTab rules have been created for a knowledge base, they are explicitly tied to it as they name particular classes and slots, which greatly hinders their reuse with further knowledge bases. We have developed a two-phase process and a supporting tool to support the reuse of JessTab rule sets. The first phase involves changing the class and slot references in the rule set into an abstract reference; the second phase involves automatically mapping between the abstract rules and further knowledge bases. Once mappings have been defined and applied for all the classes and slots in the abstract rules, the new rule set can then be run against the new knowledge base. We have satisfactorily tested our tool with several ontologies and associated rule sets; moreover, some of these tests have identified possible future improvements to the tool.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Protégé; JESS; JessTab; Rule set reuse; Ontology mapping

1. Introduction

Ontologies have become one of the most widely used forms of domain knowledge capture. When used effectively, they provide us with an explicit definition and a common understanding of a domain and the properties, relationships and behaviours of its components that can be communicated between people and machines.

RDF and RDFS [14], DAML+OIL and OWL [1] are representational formalisms for describing ontologies. Other languages provide mechanisms for querying these representations, for example RDQL [21]. Similarly, languages such as SWRL [9] and the various RuleML projects [25] allow one to formally specify rules for reasoning with the content of an ontology. While these formalisms provide a way to capture queries, an inference engine is still required to run them. One rule engine currently growing in popularity is JESS, the Java Expert System Shell; for examples of two recent projects involving JESS, see [8,12].

JESS was originally developed by the Sandia National Laboratories as a Java implementation of the popular C Language Integrated Production System (CLIPS) [3], although it has since evolved into a powerful Java rule engine and scripting language in its own right [7]. The rise in JESS's use may be in part due to the useful JessTab [5] plugin for the widely used Protégé¹ [24] ontology editor which allows developers to run JESS rules against an ontology created and populated in Protégé.

1.1. Ontology tied rules

As with CLIPS, JESS requires explicit definitions of the data types that will be used in the form of templates. Conveniently, there is a mapping command in JessTab which automatically produces these templates based on the classes and slots of a Protégé ontology. Along with the templates, JESS requires a set of facts (which are instantiations of the templates) to reason over. Again, helpfully the mapping command automatically creates facts

* Corresponding author.

E-mail address: dcorsar@csd.abdn.ac.uk (D. Corsar).

¹ Protégé-2000 and its successors up to the latest version, Protégé 3.1.

from the corresponding instances which are defined as part of a Protégé project. When writing JessTab rules, the developer refers to these templates and facts as if they had been created as part of the main JessTab program. In doing this, the rules are explicitly tied to the ontology as they are required to name particular classes and slots.

Having the rules tied to a particular ontology in this way is unavoidable, but it greatly hinders reusing a set of JessTab rules developed for one ontology with additional ontologies/knowledge bases. This is because reuse of a set of rules requires one to carry out a manual mapping between the class and slot names in the JessTab rules and those in the second (and subsequent) ontologies/knowledge bases. Further, this would be a tedious and very error prone process.

For this reason, we have developed a plugin for Protégé, which supports the developer with this task. Given a set of JessTab rules (JessTab rules differ slightly from standard JESS rules as they need to link to the Protégé ontology) and a further ontology, our tool attempts to automatically map concept names featured in the rules to concept names in the “new” ontology. To achieve this, we make use of techniques used in the ontology mapping, merging and alignment sub-fields, namely partial and exact string comparisons and synonym look-up in a lexical database (WordNet [6]). We also provide facilities for the user to define mappings manually.

In Section 2, we discuss some current ontology mapping and merging tools; in Section 3, we outline two scenarios where our tool could be used; in Section 4, we briefly outline our tool’s functionality; in Section 5, we describe experiments we have performed and report some results. In Section 6, we discuss some modifications inspired by our experiments which should improve our tool’s performance and Section 7 concludes this report with a summary of our findings.

2. Related work

As mentioned above, we make use of techniques originally developed in the ontology mapping, merging and alignment fields. There have been various approaches to these tasks including use of specially designed algebras [17], use of lexical analysis of the concept names in the two ontologies, as well as having the user manually define mappings with the aid of a specially designed user interface. Below we focus on the latter two approaches.

A popular web-based ontology management tool with the facility to assist with merging is Stanford KSL’s Chimaera [15]. Chimaera provides limited support with ontology mapping, by allowing multiple ontologies to be loaded, automatically examining their concept names and providing the user with a list of similarly named concepts. The principal mapping approach used here is the detection of common substrings. This approach is very effective for pairs of concepts that have related names, but clearly is ineffective when the same concepts are

expressed using synonyms, for example “person” and “individual”.

Another Stanford product, PROMPT [19], provides a suit of ontology management tools, in the form of a further Protégé plugin. One of these tools, iPROMPT [18], was developed as an interactive ontology-merging tool, which assists the user by providing suggestions for merging, analysing any resulting conflicts and suggesting relevant conflict resolution strategies. The latter two functions are concerned only with a pre-determined list of conflicts and are triggered in response to the user selecting one of the suggested changes.

When determining its suggested mappings, iPROMPT makes use of two factors: a measure of linguistic similarity between names, combined with the internal structure of concepts and their location in the ontology. The linguistic similarity measures of iPROMPT are based solely on substring matching of the various concept names; it makes no use of a lexical database for matching synonyms, and so consequently suffers similar problems to Chimaera.

One tool which uses lexical databases is ONION [16]. The ONION algorithm was designed to address the problem of resolving semantic heterogeneity amongst ontologies. Mitra and Wiederhold implemented two methods of addressing this task: one based, like ours, around a lexical database, while the other uses domain specific corpora. Briefly ONION’s algorithm evaluates two expressions for similarity and assigns them a similarity value. When the two terms being compared are not identical, ONION makes use of either a thesaurus (WordNet) or a corpus in assigning the value. If the value is above a user set threshold, then the mapping is accepted.

The results of Mitra and Wiederhold’s experiments indicated that generally the corpus-based mapping approach produced more accurate mappings than the thesaurus technique. However, the main difficulty with the corpus approach is in generating the corpus. In their experiments, Mitra and Wiederhold used the Google [10] search engine to find sources that were both relevant to the domain of the ontologies and contained some of their key concepts. Various sized corpora were used ranging from 50 to 1000 webpages. Although this approach provided the more accurate results, the initial investment of time and effort necessary to acquire such a corpus would no doubt be very large, making this an infeasible approach for supporting the mappings between many domains.

3. Example scenarios

There are many scenarios where reuse of an existing set of rules could be beneficial. Practically any developer building a new application requiring a set of similar or identical rules to those used in a previously developed application could benefit from the use of our tool. By semi-automating the rule reuse process, the developer could benefit from, amongst other things, reduced design, implementation and testing costs.

Below we outline two examples where our tool has been used. The first example involves building simulations of two seemingly disparate areas – shopping and water treatment, while the second emphasises the reuse of a problem solving method (ruleset) in two distinct route planning applications.

3.1. Simulating shopping and water treatment

Our first example involves the creation of two different simulations, which at their core use similar procedures, meaning their development could be achieved by using our approach. At a very abstract level, the day-to-day activity of a retail store is:

- (1) it receives stock from some supplier(s),
- (2) it stores this stock either in a storeroom or on the shelves,
- (3) customers buy items reducing the stock level,
- (4) it orders new stock,
- (5) return to step 1.

In this problem, the environment (the retail store) can be easily represented by an ontology. Likewise the processes noted above can be simulated by a series of JessTab rules coded to run against that ontology.

At a similarly abstract level, water treatment plants feature similar underlying processes:

- (1) receives water from sources,
- (2) stores it in treatment tanks,
- (3) passes the results onto desired locations,
- (4) signals that it can process more water,
- (5) return to step 1.

Again, the environment (the water treatment plant) can easily be represented by an ontology and the processes by JessTab rules. However, as the underlying processes in water treatment are in essence very similar to those of the store, our tool could be used to reconfigure the problem solver (the JessTab rules) implemented for the store to provide similar functionality for the water treatment plant.

Although it is unlikely that the second set of rules will provide all the detail required for a complete simulation of a water treatment plant, it could form the basis for such a simulation, and in so doing should reduce the development costs.

3.2. Route planning

Our previous example illustrated reuse of the same set of rules in building simulations in two different domains. Our second example is similar and illustrates the reuse of a route planning problem solver with two distinct applications, based on two different ontologies.

A highly researched field, route planning, has many applications ranging from the classical vehicle route finding

problem [4], to various tasks in robotics and artificial intelligence, to determining flow rates in IP networks [2].

Numerous algorithms have been developed over the years (for example, A*, Iterative Deepening A* (IDA*) [13], and Recursive Best First Search (RBFS)) to solve, what is essentially a search problem. A typical scenario involves finding the best path from one location to another in an environment (usually represented as a graph, in which the nodes represent states and the arcs represent pathways). The notion of “best path” can mean different things in different applications, and so algorithms typically incorporate a suitably configured evaluation metric when deciding which potential path to investigate first.

Although typically a graph is used, it is possible to represent the environment as an ontology. Classes can represent locations with attributes containing details of locations reachable from it (with classes representing pathways). Given such an ontology, a route planning algorithm could then be implemented as a series of JessTab rules to perform planning based on the ontology.

For example, a retail company could use an ontology to represent its warehouse, describing the items in it, and their locations relative to one another. A JessTab rule base could be used to calculate the shortest route around the warehouse when collecting items to fulfil orders. The same set of rules could later be reused with a second ontology representing a courier company’s deliveries and their locations. Our tool could help tailor the original planning algorithm for use against the deliveries ontology to calculate the cheapest route for the courier to take when delivering orders.

Although the above two examples are discussed at a highly abstract level, they serve to illustrate two separate uses for our tool. Both show how two tasks, which may not necessary at first appear to be related, can make use of the same underlying JessTab rules. Using our tool to achieve this reuse would simplify the process for the developer and consequentially enable him to benefit from reduced development and implementation costs for the second, and subsequent, tasks.

4. The JessTab rule reuse process

We have developed a reuse process composed of two distinct phases. The first phase consists of two actions: the extraction of class names, slot names and slot types from the relevant ontology²; and the generalisation of this information to produce a set of ontology-independent rules. During the second phase, the independent rules are mapped to the new ontology to which they are to be applied. Fig. 1 shows a representation of these processes, including the relations between the two phases. Below we discuss the two phases in more detail.

² If the relevant ontology is not available, the class and slot names could be extracted from the rule set, however the slot data type information would not be available.

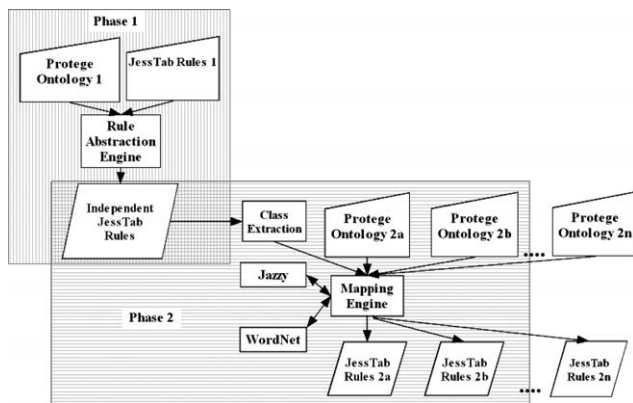


Fig. 1. Illustration of the JessTab rule reuse process.

4.1. Phase 1 – rule abstraction

The rule abstraction phase is the first stage and is identified in Fig. 1 by the vertical stripe filled box in the upper-left corner labelled “Phase 1”. During this phase, a set of JessTab rules and the corresponding ontology are passed to the Rule Abstraction Engine. Here the tool extracts from the ontology class names, and in the case of slots, both their names and their types. The JessTab rules are then rewritten replacing each class and slot name with a more general form. This removes all the references to the original ontology, making the resulting rule set more abstract and ontology-independent.

The abstract forms of both the class and slot names have a similar structure. Abstracted class names consist of a predefined prefix concatenated with the original class name. The default prefix is simply “XX_”, so, for example, every reference to the class name “Person” in the rules is replaced by the string “XX_Person”. Slots are treated similarly; however, their abstracted name consists of the prefix, concatenated with the slot name, concatenated with the slot data type. For example, an Integer slot with the name “age” is replaced by the string “XX_age_Integer”, a slot of type String with the name “title” is replaced by “XX_title_String”.

There are several reasons for transforming the names in this manner: the first is to ease the extraction of the ontological information from the rules in the second (mapping) phase; the second is to provide slot data type information to the mapping stage to enhance the automatically suggested mappings; also this naming convention is more helpful to the user than cryptic abstract names (for example, “class1”, “slot1”, “slot2”, etc.).

4.2. Phase 2 – rule to ontology mapping

After successful completion of the rule abstraction phase, the independent rules contain enough information to enable the Class Extraction component of Phase 2 to build a list of classes (and their slots). Combined with a sec-

ond ontology we have enough information to attempt to map the classes (and slots) extracted from the rules to the new ontology. This phase is illustrated by the horizontal stripe filled box in the lower right corner of Fig. 1 labelled “Phase 2”.

The basic mapping algorithm is:

- (1) Extract all the classes and their associated slot information from the abstract JessTab rules and store them in a list.
- (2) For each class extracted from the abstract JessTab rules
 - (a) Find the most suitable class in the new ontology to map to, and suggest the individual (slot and class name) mappings.
- (3) Update the rules to reflect suggested mappings and allow the user to complete and/or correct the mappings.

Clearly the key step is 2a – finding the most suitable mappings for the classes. We derive our suitability ratings by applying some commonly used lexical analysis techniques – namely string comparison, spell checking (provided by Jazzy [26]) and synonym lookup in a lexical database (WordNet accessed via JWord [11]). For a mapping to be suggested, the calculated similarity rating for two names (either class names or slot names) must exceed a threshold set by the user. Furthermore, slot mappings are only suggested for slots with the same data type. This means that if a slot extracted from the abstract JessTab rules has type String it will only be mapped to slots in the ontology of type String and not, for example, to those of type Integer or Float. This is to ensure no data type errors occur when the rules are run against the ontology. When searching for potential mappings, our tool compares every slot of every extracted class, along with the class name, with those in the ontology. If any two slots are suitable for mapping, they are given a classification based on how that mapping was achieved (the classification mechanisms are discussed below). At this level a mapping simply consists of a *from string* (the name from the abstract JessTab rules), a *to string* (the name from the new ontology) and the mapping classification. Once this process has been performed for all the classes, the mappings which have the highest values and which maintain a unique mapping between a rule-class and an ontology-class are recommended to the user as the most likely mapping for his configuration.

We define three distinct mappings operators; these are:

- (1) *Direct mapping*. Direct mappings consist of two strings that are identical, or where one has been determined to be a minor variant of the other. A typical example is the mapping from “XX_age_Integer” to the Integer slot with name “age”.

- (2) *Constituent mapping*. In constituent mappings, the constituents³ of each string are extracted and compared. If the two strings share at least a (settable) percentage of constituents, then a constituent mapping is suggested. An example is the mapping from “XX_date-of-birth_String” to a slot of type String named “birthDate” (where the threshold level is set at 60%).
- (3) *WordNet mapping*. In WordNet mappings, the constituents of each string are extracted, and WordNet is used to find synonyms of each constituent. If the percentage of synonyms in the *from string* appearing in the constituents of the *to string* (and vice versa) is greater than the threshold percentage, a WordNet mapping is suggested. A successful WordNet mapping is from “XX_yearly-earnings_Integer” to a slot of type Integer named “annual-remuneration” (where the threshold level is set at 60%), as “yearly” maps to “annual” and “earnings” maps to “remuneration” (and of course both slots are of type Integer so data type consistency is maintained).

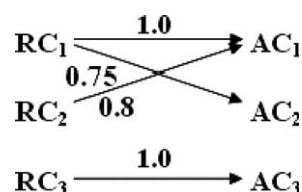


Fig. 2. Sample mappings illustrating the need for the enhanced mapping algorithm.

This mapping set is unacceptable as some of the classes are unmapped. However, our enhanced algorithm would suggest the following mappings:

$RC_1 \mapsto AC_2$ (0.75); $RC_2 \mapsto AC_1$ (0.8); $RC_3 \mapsto AC_3$ (1.0).

In this mapping set, all the classes are mapped, and if we use a simple evaluation, we would have a higher mapping score: 2 in the first case and 2.55 in the second (It is appreciated that a more sophisticated mapping function is likely to be used).

5. Results

In principle, each mapping between an extracted class and a new ontology class consists of direct, constituent and WordNet mappings. As noted earlier each of the mappings will return a value, the largest value (providing it's over a threshold) is the one which is recommended to the user. In the case of a numerical tie, the algorithm selects the mapping by applying the following precedence rule:

Direct Mappings > Constituent Mappings > WordNet Mappings

Currently the tool supports mappings between each extracted class to a single class in the ontology. In a few cases, this may result in some extracted classes not having a mapping. To minimise this, after the mapping algorithm has produced the list of mappings, a global optimisation algorithm is applied. This algorithm analyses the suggested mappings to determine if choosing a sub-optimal mapping for one class would result in more classes being mapped. If this were the case, then this alternative mapping configuration is suggested to the user.

Suppose we have a situation where we have 3 classes in the abstracted rules, let's call them RC_1 , RC_2 and RC_3 and further suppose the new ontology has 3 classes, let's call them AC_1 , AC_2 and AC_3 . Then suppose the best new mappings between the abstract rules and the actual classes are as shown in Fig. 2.

If we optimise the local mappings, we will have

$RC_1 \mapsto AC_1$ (1.0); $RC_3 \mapsto AC_3$ (1.0); and no mapping for RC_2 and AC_2 .

To evaluate the performance of our tool, we conducted a series of experiments designed to test first the rule abstraction process and second the mapping capabilities. Each test involved the development of an initial JessTab rule set, which was based around an initial ontology. This ontology was either created by ourselves or downloaded from the Stanford KSL OKBC server (accessed via the Protégé OBKB Tab plugin [22]). Once each set of rules had been developed to a satisfactory level, we built further ontologies with which to test the mapping functionality.

Tables 1⁴ and 2 provide a selection of the results from our tests with the Document ontology from the Stanford KSL OKBC server's Ontolingua section. Table 1 gives results for the mappings of selected class names and Table 2 gives the results for the slot mappings. The columns of Table 1 detail the concept name (in the Document ontology), the abstract name it was given after Phase 1, and details of the mappings produced in tests with two separate applications, detailing the concept it was mapped to and the mapping operator selected. The columns of Table 2 are identical with the additional column stating the slot type.

These results offer a good illustration of the type of support the user can expect from our tool. The “Document” concept in Table 1 demonstrates nicely the direct mapping: in Application A the name is identical, in Application B the minor spelling error in “Docment” is picked up by our tool and has no adverse effect on the suggested mapping. Both tables exhibit examples of constituent mappings, while the WordNet mappings also feature prominently both with one-word name examples (“Book” to “Volume”) and

³ The constituents of a string are the words that make up that string. We use the symbols ‘-’, ‘_’ and (in mixed case strings) uppercase letter to denote the start of new words. For example, the string *date-of-birth* has constituents *date*, *of* and *birth*.

⁴ The spelling mistakes in Table 1 are intentionally left as they illustrate the minor variant checking feature of direct mappings.

Table 1
Results for mapping the classes of the Document ontology⁴

Name	Abstract name	Mapped in to			
		Application A		Application B	
		Name	Mapping operator	Name	Mapping operator
Document	XX_Document	Document	Direct	Docment	Direct
Book	XX_Book	Book	Direct	Volume	WordNet
Thesis	XX_Thesis	Thesis	Direct	Dissertation	WordNet
Masters-Thesis	XX_Masters-Thesis	ThesisMasters	Constituent	Masters-Dissertation	WordNet
Doctoral-Thesis	XX_Doctoral-Thesis	DoctoralThesis	Constituent	Dissertation-Doctoral	WordNet
Miscellaneous-Publication	XX_Miscellaneous-Publication	AssortedPublishing	WordNet	Miscellaneous-Publishings	WordNet
Artwork	XX_Artwork	Art	WordNet	Artistry	Manual
Technical-Manual	XX_Technical-Manual	TechnicalMnual	Direct	Manual-Technical	Constituent
Computer-Program	XX_Computer-Program	ComuterProgram	Direct	Computer-Programme	WordNet
Cartographic-Map	XX_Cartographic-Map	Cartographical-Correspondence	WordNet	Cartographical-Mapping	WordNet

Table 2
Results for mapping the slots of the Document ontology

Name	Data type	Abstract name	Mapped in to			
			Application A		Application B	
			Name	Mapping operator	Name	Mapping operator
Has-Author	String	XX_Has-Author_String	Author	Direct	HasWriter	WordNet
Has-Editor	String	XX_Has-Editor_String	HasEditor	Direct	Editor-Of	Manual
Title-Of	String	XX_Title-Of_String	Title	Constituent	TitleOf	Direct
Publication-Date-Of	String	XX_Publication-Date-Of_String	IssueDate	WordNet	Date-Of-Publication	Constituent
Publisher-Of	String	XX_Publisher-Of_String	PublicationHouse	Manual	PublisherOf	Direct

multi-word concept names (“Cartographic-Map” and “CartographicalCorrespondance”).

6. Discussion

Overall we were very pleased with the outcome of these experiments. However, they do suggest that some modifications to the tool, particularly the mapping phase, could be beneficial. One significant change would be to allow the user to specify how deep the tool searches WordNet for synonyms. This is best described by means of an illustration. Take the “Artwork” concept in Table 1 line 7. In Application B, the user was required to manually map this concept to “Artistry”. When suggesting a mapping, the tool looked up synonyms of artwork in WordNet, which returned “artwork”, “art”, “graphics” and “nontextual matter”. Had the tool searched another level (i.e., looked up synonyms of art, graphics, and so on), it would have found “artistry” is a synonym of “art” and suggested a mapping. However, performing deeper searching like this could dramatically increase the time taken by the automatic mapper, and so the user would have to decide how deep to search.

A second significant change involves relaxing the slot data type matching constraint. Currently, suggested slot mappings are only made between slots of the same type (to avoid run time errors). However many slot types are compatible with each other; for example, an Integer can also be considered a Float (but not, of course, the reverse).

Relaxing this constraint would increase the number of slots considered during the automatic mapping but would still ensure no run time data incompatibility errors.

One relatively minor modification which might improve the accuracy of the automatic mapping, particularly when searching for constituent mappings, would be the removal of surplus words from concept names. Words such as “has”, “is” and “of” often feature in concept names to help the user understand their purpose. However, they have relatively little meaning for the mapping algorithm, but can often dramatically affect the similarity rating of two concept names. Consider the two concepts “Has-Author” and “Writer-Of” both used to indicate the author of a document. Currently our system would assign them a similarity rating of .50 (which, depending on the threshold level, may mean a mapping is not suggested), but by removing “Has” and “Of” we increase this to 1.0 guaranteeing a mapping is suggested.

7. Summary

The experiments performed have provided favourable results, although there are some enhancements which could be made. However, by automating part of the reuse process, our tool can be of considerable use to a developer wishing to reuse a set of JessTab rules with further ontologies. In fact, we claim to have gone some way to implementing the vision, developed by John Park and Mark Musen [20] of creating ad hoc knowledge-base systems from existing problem solving

methods (here JessTab rule sets) and knowledge bases (here Protégé OKBC/OWL ontologies).

Acknowledgements

This work is supported under the Advanced Knowledge Technologies (AKT) IRC (EPSRC Grant No. GR/N15764/01) comprising the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. See <http://www.aktors.org>.

References

- [1] G. Antoniou, F. van Harmelen, Web Ontology Language: OWL, in: Staab and Studer [23], pp. 67–92.
- [2] A.B. Bagula, Hybrid traffic engineering: the least path interference algorithm, in: SAICSIT '04: Proceedings of the 2004 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries, South African Institute for Computer Scientists and Information Technologists, 2004, pp. 89–96.
- [3] NASA/Johnson Space Center, CLIPS. <<http://www.ghg.net/clips/CLIPS.html>>.
- [4] R.D. Cuthbert, L. Peckham, APL models for operational planning of shipment routing loading and scheduling, in: WSC '73: Proceedings of the Sixth Conference on Winter simulation, ACM Press, New York, 1973, pp. 622–631.
- [5] H. Eriksson, The JESSTAB approach to Protégé and JESS integration, in: Proceedings of the IFIP 17th World Computer Congress – TC12 Stream on Intelligent Information Processing, Kluwer, B.V., Dordrecht, 2002, pp. 237–248.
- [6] C. Fellbaum (Ed.), WordNet: An Electronic Lexical Database, The MIT Press, Cumberland, RI, 1998.
- [7] E. Friedman-Hill, Jess In Action: Rule-Based Systems in Java, Manning Publications, Greenwich, CT, 2003.
- [8] C. Golbreich, Combining rule and ontology reasoners for the semantic web, in: G. Antoniou, H. Boley (Eds.), Rules and Rule Markup Languages for the Semantic Web, RuleML, Springer-Verlag, Berlin and Heidelberg, 2004, pp. 6–22.
- [9] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz, M. Deal, SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, May 2004.
- [10] Google Inc. Google. <<http://www.google.com/>>.
- [11] K. Johar, R. Simha, Jword 2.0. <<http://www.seas.gwu.edu/simhaweb/software/jword/index.html/>>.
- [12] J. Kopena, DAMLJessKB. <<http://edge.mcs.drexel.edu/assemblies/software/damljesskb/damljesskb.html/>>.
- [13] R.E. Korf, Depth-first iterative-deepening: an optimal admissible tree search, *Artif. Intell.* 27 (1985) 97–109.
- [14] B. McBride, The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS, in: Staab and Studer [23], pp. 51–66.
- [15] D.L. McGuinness, R. Fikes, J. Rice, S. Wilder, An environment for merging and testing large ontologies, in: Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000), Morgan Kaufmann, Los Altos, CA, 2000, pp. 483–493.
- [16] P. Mitra, G. Wiederhold, Resolving terminological heterogeneity in ontologies, in: Proceedings of Workshop on Ontologies and Semantic Interoperability at the 15th European Conference on Artificial Intelligence (ECAI 2002), 2002.
- [17] P. Mitra, G. Wiederhold, An ontology-composition algebra, in: Staab and Studer [23], pp. 93–116.
- [18] N.F. Noy, M.A. Musen, PROMPT: algorithm and tool for automated ontology merging and alignment, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), 2000.
- [19] N.F. Noy, M.A. Musen, The PROMPT suite: interactive tools for ontology merging and mapping, *Int. J. Hum-Comput. Stud.* 59 (6) (2003) 983–1024.
- [20] J. Park, M. Musen, The Virtual Knowledge Constructor: A Schema for Mapping Across Ontologies in Knowledge-Based Systems, PhD Thesis Proposal, Stanford University, 1999.
- [21] A. Seaborne, RDQL: A Query Language for RDF, W3C Member Submission, January 2004.
- [22] M. Sintek, OKBC Tab Website. <http://protege.stanford.edu/plugins/okbctab/okbc_tab.html/>.
- [23] S. Staab, R. Studer (Ed.), Handbook on Ontologies, International Handbooks on Information Systems, Springer, 2004.
- [24] Stanford Medical Informatics, Stanford University, Protégé Website. <<http://protege.stanford.edu/>>.
- [25] The Rule Markup Initiative, RuleML Homepage. <<http://www.ruleml.org/>>.
- [26] T. White, Can't Beat Jazzy: Introducing the Java Platform's Jazzy New Spell Checker API. <<http://www-106.ibm.com/developerworks/java/library/j-jazzy/>>.