Ontology Extraction for Distributed Environments

Derek Sleeman[1] Stephen Potter[2] Dave Robertson[2] Marco Schorlemmer[2]

[1] Department of Computing Science,
University of Aberdeen, United Kingdom
sleeman@csd.abdn.ac.uk

[2] Division of Informatics,
University of Edinburgh, United Kingdom
stephenp@aiai.ed.ac.uk, {dr,marco}@inf.ed.ac.uk

Abstract. Existing knowledge base resources have the potential to be valuable components of the Semantic Web and similar knowledge-based environments. However, from the perspective of these environments, these resources are often under-characterised, lacking the ontological characterisation that would enable them to be exploited fully. In this chapter we discuss a technique which can be applied to identify ontological knowledge implicit in a knowledge base. Based on this technique, a tool has been implemented which allows this knowledge to be extracted, thereby promoting the re-use of the resource. A discussion of some complementary research into brokering services within distributed knowledge architectures serves to illustrate the sort of environment in which such re-use might be enacted.

1 Introduction

The principal challenge for the Semantic Web community is to make machine-readable much of the material that is currently human-readable, and thereby enrich web operations from their current information-based state into a knowledge-centric form. Towards this end, for instance, the IBROW project is addressing the complex task of developing a brokering system which, given a knowledge base/knowledge source and a specification of the processing to be performed, would find an appropriate problem solver and perform any necessary transformation of the knowledge sources [1]. In this chapter, we describe one outcome of our research into techniques to enable brokering systems to become more effective and more intelligent: a technique for the semi-automatic extraction of domain ontologies from existing knowledge bases.

Work on ontologies has played a central role in recent years in Knowledge Engineering, as ontologies have increasingly come to be seen as the key to making (especially web) resources machine-readable and -processable. Systems have been implemented which help individuals and groups develop ontologies, detect inconsistencies in them, and merge two or more. Ontologies are seen as the *essence* of a knowledge base, that is, they capture, in some sense, what is commonly understood about a topic by domain experts. For a discussion of how ontologies are often developed, see [2]. Recently, systems have been implemented

which help domain experts locate domain concepts, attributes, values and relations in textual documents. These systems also often allow the domain expert to build ontologies from these entities; it has been found necessary, given the shortcomings of the particular text processed, to allow the domain expert, as part of the knowledge modelling phase, to add entities which are thought to be important, even if they are not found in the particular text [3].

Reflecting on this process has given us the insight that *knowledge bases*[1] themselves could act as sources of ontologies, as many programs essentially contain a domain ontology which although it may not be complete, is, in some sense, consistent. (Since if it were inconsistent this would lead, under the appropriate test conditions, to operational problems of the system in which the ontology is embedded). Thus, the challenge now becomes one of extracting ontologies from existing knowledge-based systems. The following section describes one approach for doing this, from, in the first instance, Prolog knowledge bases. As well as enabling their re-use, this technique can also be seen as performing a transformation of these knowledge bases into their implicit ontological knowledge.

The rest of the chapter is structured as follows. Section 2 describes, with examples, the technique for acquiring ontological knowledge from knowledge bases in a semi-automatic fashion. Section 3 gives a brief overview of our conception of a brokering system, in order to illustrate the role that this technique can play in facilitating knowledge services within distributed environments. Section 4 discusses related work, and, to conclude, Section 5 summarises the chapter.

2 Extracting Ontologies from Prolog Knowledge Bases

The method used to hypothesise ontological constraints from the source code of a knowledge base is based on Clark's completion algorithm [4]. Normally this is used to strengthen the definition of a predicate given as a set of Horn clauses, which have single implications, into a definition with double-implication clauses. Consider, for example, the predicate $member(E, L)$ which is true if E is an element of the list, L:

$$\begin{array}{l} member(X, [X|T]) \\ member(X, [H|T]) \quad \leftarrow \quad member(X, T) \end{array}$$

The Clark completion of this predicate is:

$$member(X, L) \quad \leftrightarrow \quad L = [X|T] \ \vee \ (L = [H|T] \wedge member(X, T)) \tag{1}$$

Use of this form of predicate completion allows us to hypothesise ontological constraints. For example, if we were to assert that $member(c, [a, b])$ is a true statement in some problem description then we can deduce that this is inconsistent with our use of $member$ as constrained by its completion in expression (1) above because the implication below, which is an instance of the double implication in expression (1), is not satisfiable.

$$member(c, [a, b]) \quad \rightarrow \quad [a, b] = [c|T] \ \vee \ ([a, b] = [H|T] \wedge member(c, T)) \tag{2}$$

Normally Clark's completion is used for transformation of logic programs where we are concerned to preserve the equivalence between original and transformed code. It therefore is

[1] Throughout this chapter, by "knowledge base" we mean some knowledge-bearing computer program, not necessarily expressed in some dedicated knowledge representation language, but for which the decision has been made to express the knowledge at a semantic, conceptual level. For our purposes, however, we assume that an explicit ontology describing the terms of such a knowledge base is *not* available.

applied only when we are sure that we have a complete definition for a predicate (as we had in the case of $member$). However, we can still apply it in "softer" cases where definitions are incomplete. Consider, for example, the following incomplete definition of the predicate $animal(X)$:

$$\begin{array}{rcl} animal(X) & \leftarrow & mammal(X) \\ animal(X) & \leftarrow & fish(X) \end{array}$$

Using completion as above, we could derive the constraint:

$$animal(X) \rightarrow mammal(X) \vee fish(X)$$

This constraint is over-restrictive since it asserts that animals can only be mammals or fish (and not, for instance, insects). Nevertheless, it is useful for two purposes:

- As a basis for editing a more general constraint on the use of the predicate '$animal$'. We describe a prototype extraction tool, which includes a basic editor, for these sorts of constraints in Section 2.1.
- As a record of the constraints imposed by this *particular* use of the predicate '$animal$'. We describe an automated use of constraints under this assumption in Section 2.2.

2.1 A Constraint Extraction Tool

We have produced a basic system for extracting ontological constraints of the sort described above from Prolog source code. Our tool can be applied to any standard Prolog program but is only likely to yield useful constraints for predicates which contain no control-effecting subgoals (although non-control-effecting goals such as $write$ statements are accommodated). While, in theory at least, the approach can be applied to programs of any size, we will now demonstrate the current tool using an example involving a small number of predicates.

Figure 1 shows the tool applied to a simple example of animal classification, following the introduction of the previous section. The Prolog code is:

```
animal(X) :- mammal(X).
animal(X) :- fish(X).
mammal(X) :- vertebrate(X), warm_blooded(X), milk_bearing(X).
fish(X) :- vertebrate(X), cold_blooded(X), aquatic(X), gill_breathing(X).
```

which corresponds to the Horn Clauses:

$$\begin{array}{rcl} animal(X) & \leftarrow & mammal(X) \\ animal(X) & \leftarrow & fish(X) \\ mammal(X) & \leftarrow & vertebrate(X) \wedge warm\_blooded(X) \wedge milk\_bearing(X) \\ fish(X) & \leftarrow & vertebrate(X) \wedge cold\_blooded(X) \wedge aquatic(X) \wedge gill\_breathing(X) \end{array} \quad (3)$$

The constraints extracted for this program (seen in the lower window of Figure 1) are:

$$\begin{array}{rcl} animal(X) & \rightarrow & mammal(X) \vee fish(X) \\ fish(X) & \rightarrow & vertebrate(X) \wedge cold\_blooded(X) \wedge aquatic(X) \wedge gill\_breathing(X) \\ mammal(X) & \rightarrow & vertebrate(X) \wedge warm\_blooded(X) \wedge milk\_bearing(X) \end{array} \quad (4)$$

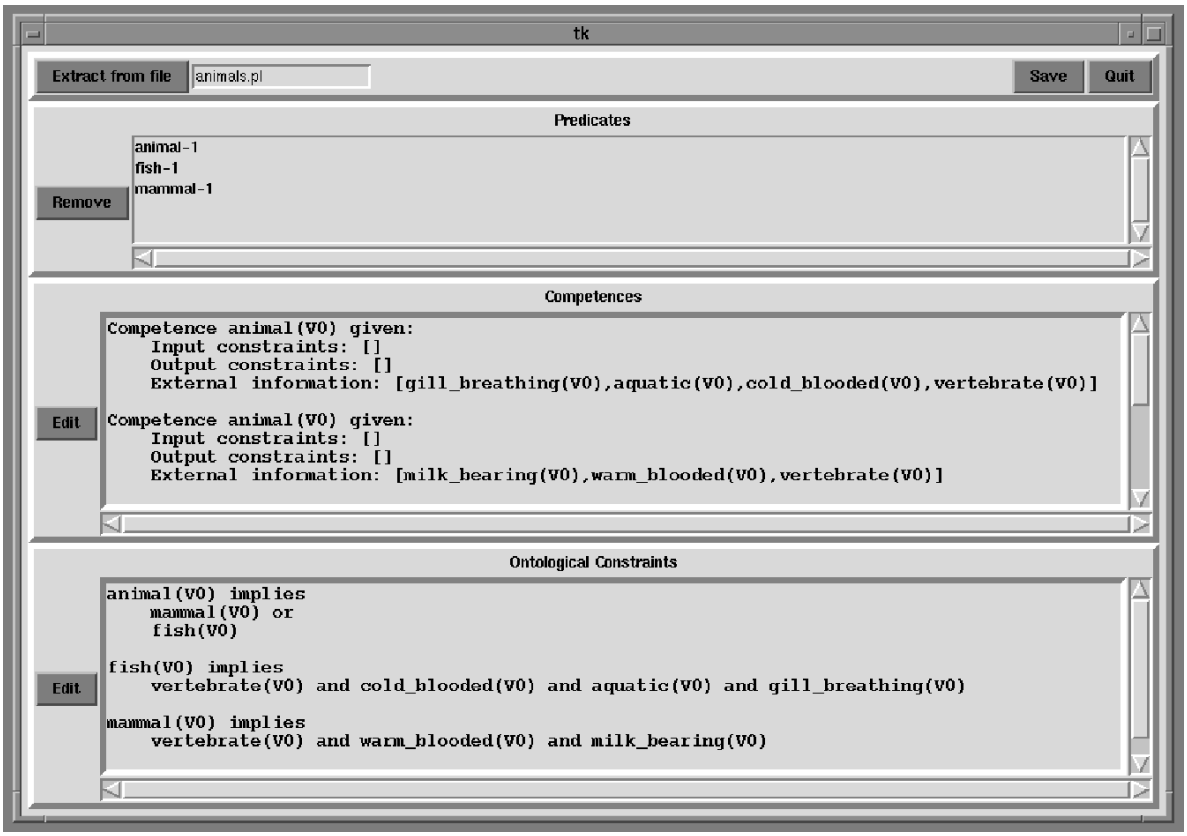

Figure 1: Ontology extraction tool

If it is deemed necessary, the user of the tool can then choose to edit manually the constraints. We show in Section 2.2 how these constraints, which, in this case, were extracted completely automatically from the Prolog source code, can be used to check another Prolog program purporting to adhere to the same ontology.

2.2 Ontological "Safe Envelopes"

The idea of running programs within ontological "safe envelopes" was introduced in [5]. Programs are run according to the normal execution control regime of the language concerned but a record is kept of the cases where the execution uses terminology which does not satisfy a given set of ontological constraints. When this happens we say the execution has strayed outside its safe envelope (from an ontological point of view). This sort of checking is not intended to alter the execution of the program in any significant way, only to pass back retrospective information about the use of terminology during an execution. This style of checking can be implemented elegantly for languages, such as Prolog, which permit meta-interpretation, allowing us to define the control structure for execution explicitly and then to augment this with appropriate envelope checking. The Horn clauses shown in expression (5) provide a basic example (extended versions of this appear in [5]).

$$\begin{array}{l} solve(true, \{\}) \\ solve((A \wedge B), E_a \cup E_b) \ \leftarrow \ solve(A, E_a) \wedge solve(B, E_b) \\ solve((A \vee B), E) \ \leftarrow \ solve(A, E) \ \vee \ solve(B, E) \\ solve(X, E \cup \{C | (X \rightarrow C \ \wedge \ not(C))\}) \ \leftarrow \ clause(X, B) \wedge solve(B, E) \end{array} \tag{5}$$

In the expressions above, $clause(X, B)$ means that there is a clause in the program satisfying goal X contingent on conditions, B (where there are no conditions, B has the value $true$). The implication $X \rightarrow C$ is an ontological constraint of the sort we are able to derive in the extraction tool of Section 2.1. The operators $\leftarrow$, $\wedge$, $\vee$, and $\cup$ are the normal logical operators for (left) implication, conjunction, disjunction and union, while $not(C)$ is the closed-world negation of condition C.

The effect of the meta-interpreter above is to test each successful goal in the proof tree for a query against the available ontological constraints. The first clause of (5) matches the goal $true$, which, as might be expected, violates no ontological constraints (and so, the empty set is returned). The second and third clauses deal with conjunctions and disjunctions of goals respectively. In the case of the former, the union of the sets of violated constraints is returned; in the latter case, the set generated by the succeeding goal is returned.

In the final clause, if an asserted `clause(X,B)` is found which satisfies the current goal, X, then the conditions, B, of this goal become subgoals of the interpreter, while the goal itself is tested against the ontological constraints. If a constraint exists ($X \rightarrow C$) that is not found to be consistent with the known facts of the current situation ($not(C)$, under the closed-world assumption), then it is added to the set of violated constraints. When a goal and its subgoals have been solved, then the interpreter exits with success, returning the set of all violated constraints; if, on the other hand, a goal cannot be solved, then the interpreter fails.

For example, suppose we have the following information about animals, `a1` and `a2`, using the animal ontology of Section 2.1.

```
animal(a1).
vertebrate(a1).
warm_blooded(a1).
milk_bearing(a1).
animal(a2).
vertebrate(a2).
cold_blooded(a2).
terrestrial(a2).
```

We could query this database in the normal way by, for example, giving the standard interpreter the goal `animal(X)` which yields solutions with `X = a1` and `X = a2`. If we want to perform the same query while checking for violations of the ontological constraints we extracted in Section 2.1, then each of these facts must now be asserted in the form (for example) `clause(animal(a1),true)`, and we pose the query via the meta-interpreter we defined above — the appropriate goal being `solve(animal(X),C)`. This will yield two solutions, as before, but each one will be accompanied by corresponding ontological constraint violations (as corresponding instances of the variable `C`). The two solutions are:

`X = a1` `C =` $\{\}$
`X = a2` `C =` $\{mammal(a2) \vee fish(a2)\}$

When presented with the first goal, `animal(a1)`, the interpreter matches this with `clause(animal(a1),true)` from the database; the precondition `true` generates no ontological problems, and from expression (4), the constraint $mammal(a1) \vee fish(a1)$ is placed on `animal(a1)`. Now, the additional facts in the database and the other ontological constraints allow the conclusion `mammal(a1)` to be drawn, so it is *not* the case that $not(mammal(a1) \vee fish(a1))$ is true (as tested by the fourth clause of the interpreter), so no constraints are violated, and the empty set is returned.

The solution of the second goal, `animal(a2)` proceeds in a similar fashion, but in this instance, the constraints and database facts do not allow either `mammal(a2)` or `fish(a2)` to be proved. Hence, under the closed-world assumption, $not(mammal(a2) \vee fish(a2))$ *is* true, and so this constraint has been violated (this in spite of the fact that the database allows the goal `animal(a2)` itself to be proved).

2.3 Extracting Ontologies from Other Sorts of Knowledge Bases

The majority of knowledge sources are not in Prolog so for our extraction tool to be widely applicable it must be able to deal with other sorts of source code. This would be very hard indeed if it were the case that the ontological constraints we extract have to encompass the entire semantics of the code. Fortunately, we are not in that position because it is sufficient to extract some of the ontological constraints from the source code — enough to give a partial match when brokering or to give a starting point for constraint editing. The issue when moving from a logic-based language, like Prolog, to a language perhaps having more procedural elements is how much of the ontological structure we can extract. We discuss this using CLIPS as an example.

Suppose we have the following CLIPS facts and rules:

```
(deftemplate person "the person template"
        (slot name)
        (slot gender (allowed-symbols female male) (default female))
        (slot pet))

(deftemplate pet "the pet template"
        (slot name)
        (slot likes))

(deffacts dating-agency-clients
        (person (name Fred) (gender male) (pet Tiddles))
        (person (name Sue) (pet Claud))
        (person (name Tom) (gender male) (pet Rover))
        (person (name Jane) (pet Squeak))
        (pet (name Tiddles) (likes Claud))
        (pet (name Claud) (likes Tiddles))
        (pet (name Rover) (likes Rover))
        (pet (name Squeak) (likes Claud)))

(defrule compatible
        (person (name ?person1) (pet ?pet1))
        (person (name ?person2) (pet ?pet2))
        (pet (name ?pet1) (likes ?pet2))
        =>
        (assert (compatible ?person1 ?person2)))
```

To extract ontological constraints from these using the current version of the extraction tool we must translate these CLIPS rules into Horn clauses. We outline below, in informal terms, the transformation algorithm needed for this task:

- For each CLIPS rule, take the assertion of the rule as the head of the Horn clause and the preconditions as the body of the clause.
- Consider each head, body or CLIPS fact as an object term.

- For each object term, refer to its `deftemplate` definition and translate it into a series of binary relations as follows:
  - Invent an identifier, I, for the instance of the object.
  - The relation $object(T, I)$ gives the type of object, T, referred to by instance I.
  - The relation $A(I, V)$ gives the value, V, for an attribute A of instance I.

  Applying this algorithm to our CLIPS example yields the Horn clauses shown below:

$$compatible(Person1, Person2) \leftarrow \begin{array}{l} object(person, O1) \wedge name(O1, Person1) \wedge pet(O1, Pet1) \wedge \\ object(person, O2) \wedge name(O2, Person2) \wedge pet(O2, Pet2) \wedge \\ object(pet, O3) \wedge name(O3, Pet1) \wedge likes(O3, Pet2) \end{array}$$

$object(person, p1)$	$name(p1, fred)$	$gender(p1, male)$	$pet(p1, tiddles)$
$object(person, p2)$	$name(p2, sue)$	$gender(p2, female)$	$pet(p2, claud)$
$object(person, p3)$	$name(p3, tom)$	$gender(p3, male)$	$pet(p3, rover)$
$object(person, p4)$	$name(p4, jane)$	$gender(p4, female)$	$pet(p4, squeak)$
$object(pet, x1)$	$name(x1, tiddles)$	$likes(x1, claud)$	
$object(pet, x2)$	$name(x2, claud)$	$likes(x2, tiddles)$	
$object(pet, x3)$	$name(x3, rover)$	$likes(x3, rover)$	
$object(pet, x4)$	$name(x4, squeak)$	$likes(x4, claud)$	

This does not capture the semantics of the original CLIPS program, since, for example, it does not express notions of state necessary to describe the operation of CLIPS working memory. It does, however, chart the main logical dependencies, which is enough for us then to produce ontological constraints directly from the extraction tool. This translation-based approach is the most direct route to constraint extraction using our current tool but we anticipate more sophisticated routes which perhaps do not translate so immediately to Horn clauses.

Extending this technique beyond knowledge representation languages to enable the extraction of ontological information from conventional procedural languages such as C would prove difficult. Programmers of these languages have no incentive to express their code at a conceptual level, with the result that the ontological constraints, insofar as they are expressed, tend to be embedded in the control elements and structure of the code to a greater extent. Code written in object-oriented languages, such as Java and C++, is potentially more susceptible to ontological extraction of this sort, since the object-oriented paradigm encourages the programmer to codify the concepts of the domain in an explicit and structured manner (the CLIPS templates in the above examples can be viewed as simple objects in this sense). However, we have yet to investigate the possibilities of mining conventional object-oriented code for ontological information.

3 Knowledge Services and Brokering

Alongside research into knowledge services such as ontology extraction, we have been pursuing parallel research into brokering mechanisms for knowledge resources (for further details see [6]). The purpose of this section is to give a brief overview of this work and to indicate how it relates to the ontology extraction tool described above (which is the principal focus of this chapter).

If the potential of the internet as a provider of knowledge-based services is to be fully realised, there would seem to be a need for automated brokering mechanisms that are able to match a customer's knowledge requirements to appropriate knowledge providers. One of the fundamental difficulties encountered when considering how to enable this sort of transaction lies in the 'semantic mismatch' between customer and provider: how should a provider advertise its services and a customer pose its queries so that advertisement and query can be matched by the broker, and the transaction successfully completed?

One possible solution to this problem, as a number of researchers into such agent-based architectures have realised (for example, see [7, 8, 9]), lies in the use of ontological knowledge. Since a well-built ontology can be seen as a conceptual 'language' expressing what is essential about a domain, and uses terms that are common to that discipline, it offers some basis for enabling communication between customer and provider. However, while there may be a large number of existing knowledge resources, not all are accompanied by explicit, machine-processable ontologies; unless some alternative approach were available, any potential gains to be made through the re-use of these resources would have to be offset against the effort involved in 'reverse-engineering' their ontologies manually. The ontology extraction tool described above in Section 2 offers one such alternative approach, by which an ontology can be constructed (semi-) automatically, thus facilitating and encouraging the reuse of knowledge.

As we conceive it, then, for the purposes of advertising its capabilities to a broker, a knowledge resource describes itself using the term:

$$k\_resource(Name, Ontology, CompetenceSet)$$

where:

- *Name* is the unique identifier of this resource;
- *Ontology* is the ontology to which the resource adheres, and by which its services can be understood, and;
- *CompetenceSet* is a set of the services, or *competences* that the resource provides and which it is making available through the broker. Each item in this set is of the form $competence(C, In, Out, G_e)$ where:
  - C is a term of the form $G \leftarrow P$, where G is a goal which is satisfiable by the resource, given the satisfaction of the conditions P.
  - In is a set of constraints placed on variables in C which must hold before the competence can be utilised (successfully).
  - Out is a set of constraints placed on variables in C which hold after the competence has been applied.
  - G_e is a set of competence goals that are known to be necessary for the successful discharge of this competence and that must be supplied by some external agent.

As should be evident, the manner in which a resource advertises its services has a major impact on the effectiveness and extent of the brokering that can be performed. We find that, although relatively concise, the above information is rich enough to allow the broker to configure complex and detailed responses to the requests it receives. When successful, these

responses are in the form of one or more brokerage structures, each describing a sequence of steps invoking the available competences of knowledge resources, which, when executed in order, should achieve the target.

Without going into too much detail about the construction of these sequences, an incoming request for service, in the form of a goal described in terms of some ontology in the system,[2] is matched against available competence-goals; the sets In, Out and G_e place additional constraints on any matches. These constraints take the form of either an ontological check of some item, or else of an additional goal that must be satisfied by the system, in which case the broker is invoked recursively. Of particular interest here is the notion of *bridges* in the system; a bridge (which will usually be constructed manually) allows terms (and thus, competences) described according to one ontology to be described according to a second ontology.[3] Bridges are a powerful concept for extending the range of the knowledge and capabilities of any system; however, they can only be defined if the ontology of a knowledge resource is made explicit.

3.1 Ontology Extraction and the Broker

It can be seen, then, that ontologies are fundamental to any approach to brokering of this sort: they enable queries to be posed to appropriate brokers, and semantic checks to be made and bridges to be built. Unfortunately, it is not realistic to expect every potential knowledge resource to be equipped with its ontology; but nor is it desirable to simply ignore those without ontologies, given the intrinsic value of knowledge resources. In this context, the extraction tool described above offers a means by which resources lacking ontological definitions can be made accessible to brokers.

This tool might be applied locally by the 'owner' of the resource in order to augment it with its ontology before introducing it into the brokering environment. Alternatively (and more interestingly), the extraction tool could itself be an agent in this environment, offering its services via the broker.

4 Related Work

In recent years there has been an increasing awareness of the potential value of ontologies — an awareness accompanied by a growing realisation of the effort required to develop them manually. As a consequence, there has been a certain amount of research into techniques by which ontological knowledge might be extracted from existing sources in which it is considered to be implicit. The aim of this section is to summarise this research, and its relationship with the ontology extraction tool described in preceding sections.

One related research area in which there has been a lot of interest, probably due to the amount of available source material, is that of ontology extraction from natural language texts. Typically, this involves identifying within a text certain linguistic or grammatical cues or patterns that suggest a certain ontological relationship between the concepts instantiating that pattern (for examples see [11, 12, 13]). Some researchers have attempted to increase

[2]Currently, it is assumed that the ontologies used to describe services are available to all. Furthermore, in this discussion, we ignore all issues of access privileges, service costs, resource management and so on that are pertinent to systems of this sort.

[3]The use of bridges here is analogous to the use of bridges in UPML[10].

the inferential power of these techniques by invoking machine learning algorithms to try to generalise the relationships that are found [14, 15]. Thus far, the successes of these text-centred approaches have been limited, with unresolved questions surrounding the extent of the background knowledge that is required for such techniques (which often try to extend an existing ontology), the amount of linguistic processing of the texts that is necessary, and, indeed, the extent and range of the ontological knowledge that it is possible to infer from texts.

Similarities can also be found to the discipline of data mining, the application of machine learning and statistical learners to large databases. As for texts, the vast numbers of data often held by organisations — and the desire to exploit these — make this an appealing approach. Applications of data mining are focused not only upon extracting ontological information, but also upon finding more 'actionable' knowledge implicit in the data. However, the limiting factor is often the data themselves: there is no guarantee that these contain any useful knowledge of any sort, but rather they are merely a collection of arbitrary or inconclusive facts. Indeed, it is often the case that the sole outcome of a data mining exercise is a confirmation of the limitations of the data in question.

The work reported here has certain parallels with the work of the software reverse-engineering community, whose members are concerned with the extraction of information from legacy software systems. There is a relationship with the *concept assignment problem* [16], the (often very difficult) task of relating program terms and constructs to the real-world entities with which they correspond. Some techniques which attempt to extract ontological knowledge from code, and which give, perhaps unsurprisingly, often mixed results, have emerged from this discipline [17, 18].

However, while our extraction tool undoubtedly has similar intentions and shares certain concerns with the work outlined above, it is distinguished from it by the choice of an existing *knowledge base* as the source of ontological knowledge. In some respects, it is surprising that hitherto there has been little research into the possibilities for extracting ontologies from such sources. In constructing a knowledge base, its developers make conscious decisions to express knowledge at a conceptual level. Consequently, it would seem to be a more immediate and more fertile ground for ontological extraction than text, data or conventional code.

5 Conclusions

The success of initiatives such as the semantic web effort will be increased if existing resources can be brought within its compass without the need for extensive re-engineering. Indeed, this might even be thought a necessary feature if these initiatives are to gain the widespread support that they require to succeed. This chapter has described a technique by which latent information — namely implicit ontological constraints — in knowledge bases can be extracted, and done so in a relatively simple, low-cost manner. This information is of the sort that enables and facilitates the future reuse and transformation of these knowledge bases within distributed environments and, as a consequence, serves to increase the scope and potential of those environments.

Acknowledgements

This work is supported under the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

References

[1] Crubezy, M., Lu, W., Motta, E., Musen, M.: The internet reasoning service: delivering configurable problem-solving components to web users. In: Proceedings of the Workshop on Interactive Tools for Knowledge Capture at the First International Conference on Knowledge Capture (K-CAP 2001), Victoria, Canada. (2001) 15–22

[2] Lopez, M., Gomez-Perez, A., Rojas-Amaya, M.: Ontology's crossed life cycle. In: Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2000), Springer (2000) 65–79

[3] Lei, G., Sleeman, D., Preece, A.: N MARKUP: a system which supports text extraction and the development of associated ontologies. Technical report, Computing Science Department, University of Aberdeen, UK (in preparation)

[4] Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Databases. Plenum Press (1978) 293–322

[5] Kalfoglou, Y., Robertson, D.: Use of formal ontologies to support error checking in specifications. In: Proceedings of the 11th European Workshop on Knowledge Acquisition, Modelling and Management (EKAW-99), Springer (1999) 207–221

[6] Schorlemmer, M., Potter, S., Robertson, D., Sleeman, D.: Knowledge Life-Cycle Management over a Distributed Architecture. Expert Update **5** (2002)

[7] Arisha, K., Eiter, T., Kraus, S., Ozcan, F., R., R., Subrahmanian, V.: IMPACT: interactive Maryland platform for agents collaborating together. IEEE Intelligent Systems Magazine **14** (2000) 64–72

[8] Nodine, M., Unruh, A.: Facilitating open communication in agent systems. In Singh, M., Rao, A., Wooldridge, M., eds.: Intelligent Agents IV: Agent Theories, Architectures, and Languages. Springer (1998) 281–296

[9] Sycara, K., Klusch, M., Widoff, S., Lu, J.: Dynamic service matchmaking among agents in open information environments. ACM SIGMOD Record (Special Issue on Semantic Interoperability in Global Information Systems) **28** (1999) 47–53

[10] Fensel, D., Benjamins, V., Motta, E., Wielinga, B.: UPML: a framework for knowledge system reuse. In: Proceedings of the International Joint Conference on AI (IJCAI-99), Stockholm, Sweden, July 31–August 5, 1999, Morgan Kaufmann (1999) 16–23

[11] Bowden, P., Halstead, P., Rose, T.: Extracting conceptual knowledge from text using explicit relation markers. In: Proceedings of the 9th European Knowledge Acquisition Workshop (EKAW-96), Nottingham, UK, May 14-17 1996, Springer (1996) 147–162

[12] Faure, D., Nédellec, C.: Knowledge acquisition of predicate argument structures from technical texts using machine learning: the system ASIUM. In: Proceedings of the 11th European Workshop on Knowledge Acquisition Modeling and Management (EKAW '99), Springer (1999) 329–334

[13] Hahn, U., Klenner, M., Schnattinger, K.: Automated knowledge acquisition meets metareasoning: incremental quality assessment of concept hypotheses during text understanding. In: Proceedings of the 9th European Knowledge Acquisition Workshop (EKAW-96), Nottingham, UK, May 14-17 1996, Springer (1996) 131–146

[14] Maedche, A., Staab, S.: Discovering conceptual relations from text. In: Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000), August 20-25 2000, Berlin, Germany, Amsterdam, IOS Press (2000) 321–325

[15] Reimer, U.: Automatic acquisition of terminological knowledge from texts. In: Proceedings of the 9th European Conference on Artificial Intelligence (ECAI-90), Stockholm, August 6-10, 1990, London, Pitman (1990) 547–549

[16] Biggerstaff, T., Mitbander, B., Webster, D.: Program understanding and the concept assignment problem. Communications of the ACM **37** (1994) 72–83

[17] Li, Y., Yang, H., Chu, W.: Clarity guided belief revision for domain knowledge recovery in legacy systems. In: Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE), Chicago, USA, Springer (2000)

[18] Yang, H., Cui, Z., O'Brien, P.: Extracting ontologies from legacy systems for understanding and reengineering. In: Proceedings of the 23rd IEEE International Conference on Computer Software and Applications (COMPSAC '99), IEEE Press (1999)