Enabling Services for Distributed Environments: Ontology Extraction and Knowledge Base Characterisation

Derek Sleeman[1], Stephen Potter[2], Dave Robertson[2], and W. Marco Schorlemmer[2]

[1] Department of Computing Science,
University of Aberdeen
sleeman@csd.abdn.ac.uk
[2] Division of Informatics,
University of Edinburgh
stephenp@aiai.ed.ac.uk, {dr,marco}@dai.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 and structural characterisation that would enable them to be exploited fully.
In this paper we discuss two currently independent services, both integrated with their environment via a brokering mechanism. The first of these services is an ontology extraction tool, which can be used to identify ontological knowledge implicit in a knowledge base. The second service involves characterising a given knowledge base in terms of the topic it addresses and the structure of its knowledge. This characterisation should permit a knowledge base to be located and assessed as a potential candidate for re-use in a more intelligent and flexible manner. The discussion of some related research into brokering systems illustrates the roles that these services can play in distributed knowledge architectures as precursors to problem-directed transformation and reuse of knowledge resources.

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]. The focus of this paper is primarily on our research into two complementary, and currently independent, techniques that enable brokering systems to become more effective and more intelligent. These techniques, then, are intended to facilitate the reuse and transformation of knowledge.

The first of these techniques involves the 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*[3] 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.

In any distributed environment, before it can be re-used or transformed, an appropriate knowledge resource must be located. Doing this efficiently is not a trivial task, since it requires the ability to identify that a resource fulfils certain domain requirements and structural criteria without entailing the need to analyse the entire content of that resource. The second technique discussed in this paper addresses this problem, attempting to summarise the essentials of a knowledge base for this particular purpose.

The rest of the paper 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 discusses the approach to characterising knowledge bases, describing them in a concise and succinct

[3] Throughout this paper, 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.

manner to better allow their re-use. Section 4 gives a brief overview of a brokering system, in order to illustrate the role that the two techniques can play in facilitating knowledge services within distributed environments. Section 5 discusses related work, and, to conclude, Section 6 summarises the paper.

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:

$$member(X, [X|T])$$
$$member(X, [H|T]) \leftarrow member(X, T)$$

The Clark completion of this predicate is:

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

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.

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

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 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 $animal(X)$:

$$animal(X) \leftarrow mammal(X)$$
$$animal(X) \leftarrow fish(X)$$

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: the *EXTRACTexP* System

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:

(3)
$$animal(X) \leftarrow mammal(X)$$
$$animal(X) \leftarrow fish(X)$$
$$mammal(X) \leftarrow vertebrate(X) \wedge warm\_blooded(X) \wedge milk\_bearing(X)$$
$$fish(X) \leftarrow vertebrate(X) \wedge cold\_blooded(X) \wedge aquatic(X) \wedge gill\_breathing(X)$$

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

(4)
$$animal(X) \rightarrow mammal(X) \vee fish(X)$$
$$fish(X) \rightarrow vertebrate(X) \wedge cold\_blooded(X) \wedge aquatic(X) \wedge gill\_breathing(X)$$
$$mammal(X) \rightarrow vertebrate(X) \wedge warm\_blooded(X) \wedge milk\_bearing(X)$$

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

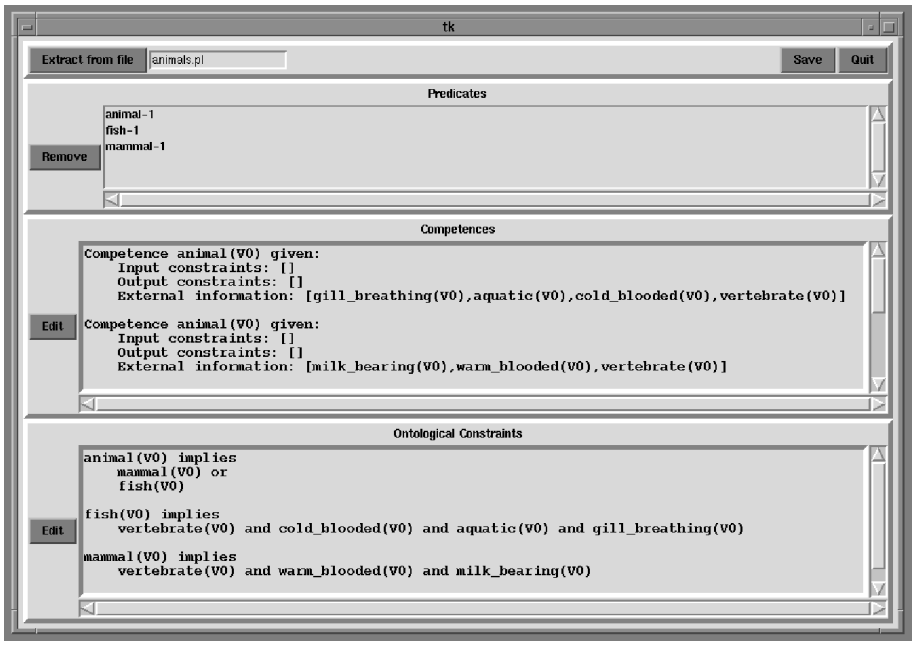

Fig. 1. Ontology extraction tool

in expression (5) provide a basic example (extended versions of this appear in [5]).

(5)

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

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, for example by giving 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 is asserted in the form, e.g., `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 EXTRACTexP 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:

$$compatible(Person1, Person2) \leftarrow$$

$$object(person, O1) \wedge name(O1, Person1) \wedge pet(O1, Pet1) \wedge$$
$$object(person, O2) \wedge name(O2, Person2) \wedge pet(O2, Pet2) \wedge$$
$$object(pet, O3) \wedge name(O3, Pet1) \wedge likes(O3, Pet2)$$

$$\begin{array}{llll} object(person, p1) & name(p1, fred) & gender(p1, male) & pet(p1, tiddles) \\ object(person, p2) & name(p2, sue) & gender(p2, female) & pet(p2, claud) \\ object(person, p3) & name(p3, tom) & gender(p3, male) & pet(p3, rover) \\ object(person, p4) & name(p4, jane) & gender(p4, female) & pet(p4, squeak) \end{array}$$

$$\begin{array}{ll} object(pet, x1) & name(x1, tiddles) \; likes(x1, claud) \\ object(pet, x2) & name(x2, claud) \quad likes(x2, tiddles) \\ object(pet, x3) & name(x3, rover) \quad likes(x3, rover) \\ object(pet, x4) & name(x4, squeak) \; likes(x4, claud) \end{array}$$

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 EXTRACTexP. 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 Characterising Knowledge Sources in a Distributed Environment

Reuse of both problem solving components and knowledge sources is a holy grail of knowledge engineering. While there has been considerable discussion of re-use of problem solving algorithms in Knowledge Engineering [6] and in Software Engineering [7], there has been much less work on reuse of knowledge bases/sources. But if a company has spent a great deal of time and resource in developing a knowledge base for, say, the design of an engine, it would seem prudent, if it were possible, to use that same knowledge as the basis for a diagnostic knowledge base. In general one could imagine designers making such requests over the internet/company intranet:

> "I am looking for a knowledge base which discusses the design specification of machines for grape harvesting."

In general, these requests can be characterised as "Require knowledge base on topic T" or, more likely, "Require knowledge base on topic T where the knowledge conforms to certain constraints C". The ability to respond to a request of this form would be an important step towards creating the sort of environment in which the re-use of knowledge components is a commonplace.

We plan to address such knowledge base characterisation issues as follows: Firstly, we will decide what is the principal topic, T, of a given knowledge base. Secondly we will develop a series of other programs (or *filters* if one uses a LARKS-like nomenclature [8]) to look for different kinds of structure/constraints in the knowledge base. Each of these is dealt with briefly below.

3.1 Knowledge Base Topic Identification

Our current EXTRACTexP system can analyse a Prolog knowledge base, and can extract all the predicates (and their arities) which it contains (see top window of the tool in Figure 1). Using knowledge of Prolog, its basic constructs like `read`, `write`, etc. are discarded, leaving a set of domain terms. These terms could then be propagated through a predefined ontology (*c.f.* the spreading activation through a Semantic Network which was postulated in the '70s as a possible model to explain human focus of attention change [9]). This ontology would contain relevant concepts within the universe of discourse.

As a simple example, suppose we have the ontology depicted in Figure 2. If the concepts `Apples` and `Pears` were passed to the system, it would suggest that `Fruit` might be the relevant focus of the knowledge base. Similarly, providing the set `{Apples, Pears, Potatoes, Carrots}` would suggest that `Fruit-Vegetables` might be the focus, and if one provided `{Apples, Potatoes, Chicken, Game}` it would suggest `Food` might be the focus. We plan subsequently to extend the system so that it will be able to detect two or more principal topics, *e.g.* `Fruit` and `Food Processing`, drawn from a number of complementary ontologies.

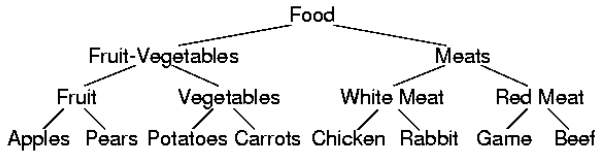

Fig. 2. Example classification ontology

3.2 Knowledge Base Structure Filters

Filters which detect the structure of a knowledge source might:

- Constrain knowledge sources such that a high percentage of their knowledge elements contain entities from both ontologies. So using the example from the last point, an appropriate goal for filtering might be "$> P$% of elements would contain elements from the Fruit/Food ontology and the Food Processing ontologies".
- Require that elements of the knowledge base be strongly related. Earlier in the COCKATOO system we demonstrated that we could acquire knowledge bases/data sets which were essentially consistent with an Extended BNF grammar [10]. Here, with the 'essentials' of the required knowledge expressed through such a grammar, rather than using this approach to acquire a knowledge base conforming to that grammar, it can instead be used to check whether existing knowledge resources display an acceptable degree of coherence with respect to the grammar. To enable such an approach, it is likely that the elements of the knowledge source would need to be marked up in XML or some comparable notation. As an illustration, below we give a section of such an EBNF grammar we used in the earlier work to describe rock formations [10]:

```
formation     → <lithology>+
lithology     → (<rock> <lithology-depth>
                              [<lithology-length>])
rock          → (<rock-type> <rock-hardness>)
rock-type     → (shale | clay | chalk | granite
                                          | other)
rock-hardness → (very-soft | soft | medium | hard
                                       | very-hard)
```

4 Knowledge Services and Brokering

In work reported elsewhere ([11]), we have been pursuing parallel research into brokering mechanisms for knowledge resources; the purpose of this section is to give a brief overview of this work and to indicate how it relates to the knowledge services described above which are the principal focus of this paper.

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 [12,13,8]), 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,[4] 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.[5] 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.

[4] 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.

[5] The use of bridges here is analogous to the use of bridges in UPML[6].

4.1 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, EXTRACTexP offers a means by which resources lacking ontological definitions can be made accessible to brokers.

4.2 Knowledge Base Characterisation and the Broker

While this should lead to more flexible and intelligent environment, the language available for expressing queries to the broker is still relatively impoverished, and perhaps not best suited to the sort of queries that will arise in a knowledge-centred system. In particular, while a certain knowledge resource may conform to a particular ontology and satisfy stated goals consistent with that ontology, this gives little indication of the range of the knowledge base, and the structure of the inferences it can make. To address this problem, we can call upon the knowledge base characterisation services outlined above in Section 3.

Consider the case where a query to the broker is now in the form of a request for a knowledge resource that addresses a topic T, and which conforms to a set of constraints C. The technique outlined in Section 3.1 allows a description of the topic that the resource addresses to be extracted. This description is in terms of some pre-defined ontologies that could be supplied by the querying agent. Alternatively (and perhaps more appropriately) these ontologies could be managed by the broker itself, along with any known characterisations of available knowledge sources. The topics of potential candidate resources known to the environment could be extracted by the tool (again acting as a knowledge-service provider in this environment) at the instigation of the broker.

Assuming that a number of knowledge resources are found that cover the desired topic area, the next step would be to apply the constraint set C to these candidate resources, through the invocation (in a similar fashion) of the appropriate filters. The result of this would be to locate those resources (if any) that match the initial query. While providing no guarantee that these will fulfil the querying agent's needs, it would seem to offer an approach that goes beyond the simple syntactic matching often adopted, and a move towards richer, semantic modes of transaction.

5 Related Work

The aim of this section is to summarise the related work in the fields of ontology extraction and knowledge base characterisation and, as a result, set the knowledge services described in this paper in their proper context.

5.1 Ontology Extraction

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 promising 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 [14,15,16]). Some researchers have attempted to increase the inferential power of these techniques by invoking machine learning algorithms to try to generalise the relationships that are found [17,18]. 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* [19], 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 [20,21].

However, while the EXTRACTexP tool undoubtedly has similar intentions and shares certain concerns with the work outlined above, it is distinguished from them 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.

5.2 Knowledge Base Characterisation

The characterisation of knowledge bases (and, more generally, knowledge-based systems) has been a concern of AI research for many years, principally for the purposes of construction and analysis. As one example, the KADS [22] methodology involves characterising the knowledge surrounding a particular task — and hence, the knowledge base to address that task — at *task*, *inference* and *domain* levels according to its nature and content, and the role that it plays in problem-solving. This sort of characterisation can promote the re-use of knowledge components (of, for example, problem-solving methods at the inference level). More recently, projects such as that to develop UPML [6] have extended some of these ideas with the express purpose of modelling distributed environments of knowledge-based components.

Here, we are interested specifically in characterising a knowledge base from the perspective of the potential re-user, and the nature of the requests for knowledge that are made. However, a feature essential to the work reported here is that, if it is to be of use, this characterisation should be (at least) semi-automatic; there has been little work published regarding this aspect, and, as such, we believe that there is a contribution to be made in this area.

6 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 paper has introduced two techniques that, in a relatively simple, low-cost manner, extract latent information from knowledge bases, namely implicit ontological constraints and characterisation information. 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. M. Crubezy, W. Lu, E. Motta, and M. A. Musen. The internet reasoning service: delivering configurable problem-solving components to web users. In *Proc. Workshop on Interactive Tools for Knowledge Capture at the First International Conference on Knowledge Capture (K-CAP 2001), Victoria, Canada*, pp. 15–22, 2001.

2. M.F. Lopez, A. Gomez-Perez, and M.D. Rojas-Amaya. Ontology's crossed life cycle. *Proc. EKAW-2000 Conference, Juanles-Pins, France*, Springer, pp. 65–79, 2000.
3. G. Lei, D. Sleeman, and A. Preece. 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. K.L. Clark. Negation as failure. In H. Gallaire, and J. Minker (eds.), *Logic and Databases*, pp.293–322. Plenum Press, 1978.
5. Y. Kalfoglou and D. Robertson. Use of formal ontologies to support error checking in specifications. In *Proc. 11th European Workshop on Knowledge Acquisition, Modelling and Management (EKAW-99), Germany*, pages 207–221. Springer Verlag (Lecture Notes in Computer Science 1621), 1999.
6. D. Fensel, V.R. Benjamins, E. Motta and B. Wielinga. UPML: a framework for knowledge system reuse. In *Proc. International Joint Conference on AI (IJCAI-99), Stockholm, Sweden, July 31–August 5, 1999*, Morgan Kaufmann, pp. 16–23, 1999.
7. D.J. Reifer. *Practical Software Reuse*. John Wiley, New York, 1997.
8. K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic service matchmaking among agents in open information environments. In *ACM SIGMOD Record (Special Issue on Semantic Interoperability in Global Information Systems)*, A. Ouksel, A. Sheth (Eds.), **28**(1), March 1999, pp. 47–53, 1999.
9. A.M. Collins, and E.F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, **82**, pp. 407–428, 1975.
10. S. White, and D. Sleeman. A grammar-driven knowledge acquisition tool that incorporates constraint propagation. In *Proc. First Int Conf on Knowledge Capture (KCAP-01), October 21–23, Victoria, Canada*, ACM Press, pp. 187–193, 2001.
11. W. M. Schorlemmer, S. Potter, D. Robertson, and D. Sleeman. Formal knowledge management in distributed environments. In Workshop on Knowledge Transformation for the Semantic Web, 15th European Conference on Artificial Intelligence *ECAI-2002*, Lyon, France, 2002.
12. K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross and V.S. Subrahmanian. IMPACT: interactive Maryland platform for agents collaborating together, *IEEE Intelligent Systems magazine*, **14**(2), pp. 64–72, 2000.
13. M. Nodine, and A. Unruh. Facilitating open communication in agent systems. In *Intelligent Agents IV: Agent Theories, Architectures, and Languages*, M. Singh, A. Rao and M. Wooldridge (Eds.), pp. 281–296. Springer-Verlag (Lecture Notes in AI V. 1365), 1998.
14. P.R. Bowden, P. Halstead, and T.G. Rose. Extracting conceptual knowledge from text using explicit relation markers. In N. Shadbolt, K. O'Hara, and G. Schreiber (eds.), *Proc. Ninth European Knowledge Acquisition Workshop (EKAW-96), Nottingham, UK, May 14-17 1996*, Springer-Verlag, Berlin, pp. 147–162, 1996.
15. D. Faure, and C. Nédellec. Knowledge acquisition of predicate argument structures from technical texts using machine learning: the system ASIUM. In D. Fensel, R. Studer (eds.), *Knowledge Acquisition, Modeling and Management, Proc. Eleventh European Workshop, EKAW '99, Dagstuhl Castle, Germany, May 26-29, 1999* Lecture Notes in Computer Science, Vol. 1621, Springer, Berlin. pp. 329–334, 1999.
16. U. Hahn, M. Klenner, and K. Schnattinger. Automated knowledge acquisition meets metareasoning: incremental quality assessment of concept hypotheses during text understanding. In N. Shadbolt, K. O'Hara, and G. Schreiber, (eds.), *Proc. Ninth European Knowledge Acquisition Workshop (EKAW-96), Nottingham, UK, May 14-17 1996*, Springer-Verlag, Berlin, pp. 131–146, 1996.
17. A. Mädche, and S. Staab. Discovering conceptual relations from text. In W. Horn (ed.), *Proc. Fourteenth European Conference on Artificial Intelligence (ECAI 2000), August 20-25 2000, Berlin, Germany*, IOS Press, Amsterdam, pp. 321–325, 2000.
18. U. Reimer. Automatic acquisition of terminological knowledge from texts. In L. C. Aiello (ed.), *Proc. Ninth European Conference on Artificial Intelligence (ECAI-90), Stockholm, August 6-10, 1990*, Pitman, London, pp. 547–549, 1990.
19. T.J. Biggerstaff, B.G. Mitbander, and D.E. Webster, Program understanding and the concept assignment problem," *Comm. ACM*, **37**(5), pp. 72–83, 1994.
20. Y. Li, H. Yang, and W. Chu. Clarity guided belief revision for domain knowledge recovery in legacy systems. In *Proc. 12th International Conference on Software Engineering and Knowledge Engineering (SEKE), Chicago, USA*, Springer, 2000.
21. H. Yang, Z. Cui, and P. O'Brien. Extracting ontologies from legacy systems for understanding and re-engineering. In *Proc. IEEE 23rd International Conference on Computer Software and Applications (COMPSAC '99), October 1999*, IEEE Press, 1999.
22. G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N.R. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge Engineering and Management*, MIT Press, 2000.