Knowledge Engineering with Software Agents

Louise Crow & Nigel Shadbolt

Artificial Intelligence Group,
Department of Psychology,
University of Nottingham,
University Park, Nottingham
NG7 2RD
U.K.
+44 (0) 115 9515280
lrc@psychology.nottingham.ac.uk, nrs@psychology.nottingham.ac.uk

Abstract

Increasingly diverse and useful information repositories are being made available over the World Wide Web (Web). However, information retrieved from the Web is often of limited use for problem solving because it lacks task-relevance, structure and context. This research draws on software agency as a medium through which model-driven knowledge engineering techniques can be applied to the Web. The IMPS (Internet-based Multi-agent Problem Solving) architecture described here involves software agents that can conduct structured on-line knowledge acquisition using distributed knowledge sources. Agent-generated domain ontologies are used to guide a flexible system of autonomous agents arranged in a server architecture. Generic problem solving methods developed within the expert system community supply structure and context.

1 Knowledge Engineering and the Web

Knowledge engineering is concerned with the development of knowledge-based (expert) systems to perform tasks. In knowledge engineering, the 'knowledge acquisition bottleneck' refers to the time- and resource-intensive process of getting the knowledge about a domain that is necessary to perform the required task. Some examples of domains (subject areas) are respiratory medicine, igneous petrology and electronic engineering. Knowledge engineers acquire knowledge of the domain from various sources and represent it in a form that can be used in a knowledge based system. It is particularly difficult to get access to human domain experts who are, by definition, rare and often very busy. Therefore it is usual to perform initial sessions of domain structuring and representation using available resources such as textbooks, manuals, and increasingly, electronic media.

While the Web could be a valuable source of knowledge covering many subject areas, it also presents the problem that there is often simply *too much* information available to each Web user. Relevant facts are obscured by masses of irrelevant data. An analysis of the Web using the overlap between pairs of search engines to estimate its size gave a lower bound on the size of the "indexable Web" in December 1997 of 320 million pages (Lawrence and Giles 1998). This may be a serious underestimate of the size of the whole Web as search engines typically do not index documents hidden behind search forms and cannot index some documents, which use the "robot exclusion standard" or require authentication. The estimate also excludes dynamic on-the-fly information serving over the Web. This exclusion is significant because increasingly, the Web is being used as a gateway for dynamic information transfer rather than simple delivery of static HTML pages. Programs are interfaced (e.g. through CGI-bin scripts) to the Web to provide users with information compiled on-the-fly and tailored to their needs. Even the static content of the Web is continually changing. Information is constantly updated, links are changed or expire, and pages (or entire Web sites) are moved to new locations in virtual and real space.

There are other factors to be considered when using information from the Web. It holds information in many different formats. Examples include institutional databases, FAQ lists, interactive programs, publications, statistics, descriptive texts, photographs, films, sound and animation. A knowledge engineer will usually wish to exploit more than one source of information. The use of multiple formats and the distributed nature of the Web make the integration of this information a non-trivial task. Firstly each piece of information must be retrieved from its location using network protocols, and re-represented in a common format. Secondly, the pieces of information must be integrated with respect to their meaning. This poses problems as information from different sources may have been created from different, possibly contradictory, perspectives on the subject concerned.

Although the Web has become well established and influential, we are only just beginning to explore the potential of tools that retrieve collated information from it. The ability to filter and reassemble information in an appropriate structure and context will be increasingly important as more and more information becomes available on-line. This paper presents the IMPS (Internet-based Multi-agent Problem Solving) architecture. IMPS is an agent-based architecture driven by knowledge level models. It is designed to facilitate the retrieval and restructuring of information from the Web. Our purpose is to use the resulting knowledge as problem solving knowledge suitable for use in a knowledge based system. However, the approach could have wider applicability.

A semantic distinction can be made at this point between information and knowledge. Knowledge can be thought of as the whole body of information that someone brings to bear to solve a problem. One salient feature of this definition of knowledge is the composition of individual items of information into a larger pattern. The other is the application of this information to reach a goal. The advent of the Internet has increased the amount of available electronic information. The Web holds information on a vast number of domains. However, it does not present convenient packages of domain knowledge indexed by the kind of problems to which they could be applied. Within each domain represented on the Web, information is available which could be used for solving many possible problems. IMPS uses an approach that extracts and transforms information based on two criteria. The approach considers firstly the subject or domain (e.g. geology, respiratory medicine, electronic engineering), and secondly the kind of task in which the information is to be used (e.g. classification of objects, diagnosis of faults in a system, scheduling a series of tasks). In effect, this approach describes on-line knowledge acquisition using knowledge level models.

1.1 Knowledge Level Models

Knowledge level models are widely used for knowledge acquisition in the knowledge engineering community. The term harks back to Newell's original distinction between the knowledge level and the symbol level (Newell 1982). These are ways of interpreting the behaviour of any agent seen by an external observer. The behavioral law used to understand the agent at the knowledge level is the principle of maximum rationality which states: "If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action". No further attempt is made at this level to delve into how the agent actually works. It is enough to know that it acts as if according to the principle of maximum rationality. The knowledge level is implemented by the symbol level. At the symbol level, the agent is seen as a mechanism acting over symbols, and its behaviour is described in these terms.

This distinction has had a great influence on the knowledge engineering community, notably in the modeling approaches to building knowledge based systems. Here, the systems are viewed as models of expert problem solving behaviour. Using the knowledge level, this behaviour can be modeled in terms of knowledge content (i.e. epistemologically).

However, while Newell's conceptualization did not allow for any structure at the knowledge level, most of the knowledge level models (KL-models) used in knowledge based systems are highly structured (Van de Velde 1993). Different kinds of knowledge model generally represent a particular perspective on the knowledge level. Van de Velde points out that in practical terms, knowledge can be divided into relatively stable structures.

> A KL-model is a structure that is imposed on knowledge when it is being put to use in a class of problem situations

The common feature of this class of problem situations may be that they are all in the same domain. In this case, the model imposed may be described as a domain model. This is a means of expressing domain knowledge in a precise, systematic way. For example, if the domain being considered was respiratory diseases, a domain model might contain knowledge about the structure and function of a pair of lungs. This model could be useful in various tasks within the domain. It could facilitate diagnosis of respiratory disease by providing a causal model for linking diseases to symptoms, or it could be used to classify respiratory diseases by their features.

Alternatively, the common feature of the class of problem situations might be that they share a common task structure, although they appear in different domains. This would be true of classification problems in geology and electronic engineering. Say the problems were a) to classify an unknown igneous rock specimen b) to classify an unknown circuit component. In both cases, a knowledge model might be used which represents knowledge about the classes that the unknown object may belong to, the distinguishing attributes of those classes, and the features of the unknown object which may be matched against those attributes. This kind of model is known as a task model. There may be more than one way to solve problems that share a task model. For instance, there are many problem solving methods for carrying out classification tasks.

A full knowledge level model would bring together the task model, domain model and problem solving method into a coherent model that is sufficient to solve the task. If this is the case, one may ask "Why use the separate models in the first place?". One of the reasons that these different knowledge models are used is to guide knowledge acquisition. If a new domain is being tackled, a task model may impose a structure on knowledge that can guide acquisition, even though a full domain model has not yet been formed. For instance, in the case of rock classification, although nothing may be known about the features of rock, a knowledge engineer will know that they must try to acquire information about distinguishing properties of classes of rock.

Another reason for using the separate models is the potential for knowledge reuse. Full knowledge level models will rarely be reused, as it is unlikely that knowledge engineers will encounter exactly the same problem again. However, domain models, task models and problem solving methods all have the potential to be reused in problems which are similar in domain (in the case of domain models), or task (in the case of task models and problem solving methods).

1.1.1 Problem Solving Methods Reusable problem-solving methods (PSMs) focus on the idea that certain kinds of common task (e.g. classification) can be tackled by using the same problem-solving behaviour (e.g. generate and test a set of hypotheses), regardless of the domain in which they appear. An abstract model of this behaviour is a problem solving method. This method relates the task and domain models together so that goals can be accomplished. Knowledge roles (e.g. object class, attribute) in the task model are instantiated by domain knowledge. The problem solving method performs transformations between the input roles of the task model (i.e. the information that is given to the problem solver) and the output roles (the solution which is expected of it).

The separation of a problem solving method from domain knowledge has been used in various well-known knowledge acquisition methodologies (Weilinga et al. 1991) (Klinker et al. 1991) (Steels 1990) (Chandrasekaran 1986). What these approaches have in common is the use of a finite library of domain independent problem solving methods, which may need some tuning to suit the domain of application.

Some methods, such as ‘generate and test’ or ‘heuristic search’ are so weak in their assumptions of domain knowledge that they can be applied to a wide range of tasks with little fine-tuning. However, these weak methods tend not to be very efficient. Stronger, more efficient methods have more requirements in terms of the type and structure of domain knowledge and are therefore less widely applicable (Bylander and Chandrasekaran 1988). Figure 1 shows a typical domain knowledge schema for classification.

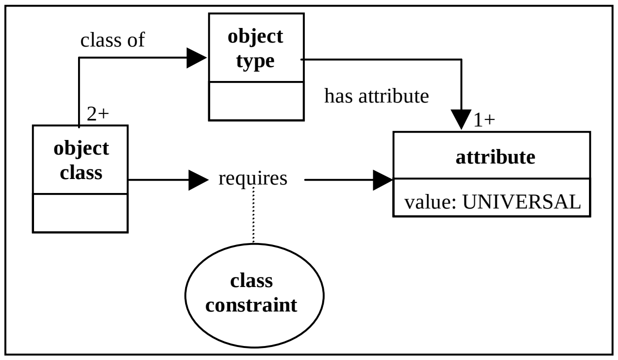


Figure 1: Typical domain-knowledge schema for classification tasks (Schreiber et al. 1998)

Additionally, problem solving methods are usually specified not as a homogenous whole, but as a series of components or inference steps. Each of these components describes a relatively independent step taken in the problem solving method. Each oval in Figure 2 represents an inference step taken in the pruning classification method. There is often some flexibility regarding the order these steps are taken in. The term grainsize refers to the size of these elemental components.

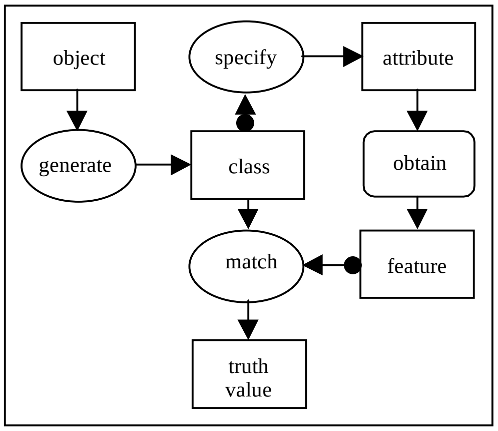


Figure 2: Inference structure for the pruning classification method (Schreiber et al. 1998)

Methods with larger grainsize – fewer and larger components – are less reusable and require more tuning for new uses. Therefore the approach is moving towards a smaller grainsize, with finer-grained problem-solving strategies which can be configured together to form a knowledge based system (Puerta et al. 1992) (Gil and Melz 1996). Knowledge based system metatools have appeared (Walther, Eriksson and Musen 1992) (Studer et al. 1996) which store reusable components in libraries and configure them according to the requirements of each application.

Recently, researchers have begun to discuss the implications of the Internet for problem solving method reuse in particular and knowledge engineering in general (e.g. Benjamins 1997). Fensel (Fensel 1997) points out that one reason that actual reuse of PSMs hasn't occurred as often as it might is that the PSMs that have been implemented make strong assumptions on software and hardware environments that limit reuse in other environments. Although a PSM itself may be specified at the knowledge level, the symbol level implementation of these knowledge level models on a computer is a practical impediment to reuse. It is difficult to reuse a PSM that has been implemented in Lisp on a Mac in a system using C++ on a Unix machine although they may be compatible at the knowledge level. These kinds of limitations are bypassed in an Internet-based architecture, such as IMPS, which exploits the emerging open standards for interoperation in Web-based software, such as the platform-independent Java language.

1.1.2 Ontologies As we have discussed, it is the conjunction of the domain and task models with the problem solving method that allows a knowledge based system to achieve goals. The idea of a library of domain independent problem solving components, such as task models and problem solving methods implies the availability of domain models to instantiate these components and turn them into knowledge based systems.

Domain models can also be known as domain ontologies. It has been argued (van Heijst, Schreiber and Wielinga 1997) that explicit ontologies can be used to underpin the knowledge engineering process. However, the concept 'domain ontology' is used ambiguously in the knowledge engineering community (Uschold 1998). It can mean:

> …the particular subject matter of interest in some context…The nature and scope of a domain are determined by what is of interest and the context. The context includes the purpose for delimiting a subject area.

An alternative perspective (also taken from Uschold) sees the ontology covering

> the particular subject matter of interest in some context *considered separately* from the problems or tasks that may arise relevant to the subject.

According to the first definition, an ontology can be task dependent – so a domain ontology might only contain concepts and relationships relevant to the particular problem solving method with which it was used. An example of this might be an ontology of rocks to be used in a simple classification task. This could contain the mineral properties of different types of rock necessary for identifying them, but would not necessarily contain information about the causal processes that contributed to rock formation. These causal concepts would be relevant to a different problem solving method in which some causal model of process was used.

The second conceptualization proposes that a domain ontology should be constructed with no reference to the kind of tasks that the knowledge engineer would be hoping to tackle with it. This kind of domain ontology in the field of geology would contain all the information available about rocks. It has been suggested that a library of such task-independent domain ontologies would complement the existing libraries of task models and problem solving methods. In Figure 3 it can be seen that while a reusable problem solving method should be situated in area C (which contains abstract task-relevant knowledge), a domain ontology could be positioned either in area A (if the first definition is used) or area B (if the second definition is used).

We have discussed how problem solving methods can make assumptions about the nature and structure of the domain knowledge that can be used with them. Therefore, if a domain ontology is not oriented towards the specific task to be carried out in the domain (i.e. it is situated in area B), there may have to be extensive modification of the information in the domain ontology, or 'mapping', to allow it (and subsequent knowledge acquisition based on the ontology) to be fitted into the constraints of the problem solving model. This process is much simpler if the domain ontology is geared towards the task from the outset – a situation that is difficult to achieve with a library of reusable domain ontologies.

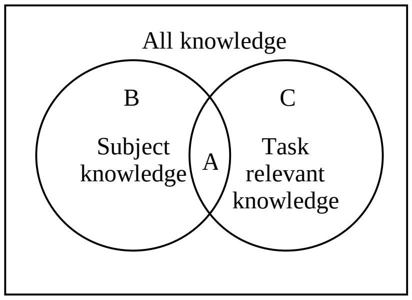


Figure 3: How knowledge can be separated into knowledge level models

Other significant obstacles would need to be overcome in the creation of such a library (van Heijst, Schreiber and Wielinga 1997). In order to provide usable ontologies for a significant range of domain areas, the library itself would have to be huge. In order to make a system useful in areas not covered by the library, some method for supplying ontologies to 'cover the gaps' would be required. Both this problem and the interaction between PSMs and domain knowledge seem to indicate that an alternative to the idea of ontology libraries might be appropriate. Our hypothesis is that more use and reuse will be obtained from a system that constructs ontologies at runtime from some set of source material, fitting the ontologies to the requirements of the problem solving method being used.

Such an approach is also an asset in dealing with a changing information environment in which an adequate ontology of domain terms two years ago may no longer be adequate today. Tennison and Shadbolt (Tennison and Shadbolt 1998) argue for a move to "living ontologies" whose development is integrated with that of the system they are to be used in. A system that can integrate and use knowledge from different sources to construct a domain-specific, task-specific ontology could be used both to create new ontologies for domains, and also to update existing ontologies, or adapt ontologies created for different tasks.

The idea of generating domain ontologies was explored in the SENSUS project (Swartout et al. 1996). This involved the use of a broad coverage general ontology to develop a specialized, domain specific ontology semi-automatically. The availability of high level ontologies on the Internet is increasing. These high level ontologies are attempts to produce a domain model for the world. Systems such as the Generalized Upper Model (Bateman, Magnini and Fabris 1995), the 'upper CYC® ontology' (Cycorp Inc. 1997), Ontolingua (Gruber 1993) and

WordNet (Miller 1990) are knowledge structures which provide a framework which could organize all of the concepts we use to describe the world, aiming to cover every possible subject area with at least a low level of detail. They are generally hierarchical and carve up the world into a set of high level classes. For example, 'event' and 'entity' are top level classes in WordNet. It should be possible, in theory, to position any concept somewhere in the hierarchy of increasingly specific terms that springs from these high level concepts. These ontologies may be used to as a 'bootstrapping' method to bridge the gap between "common sense" knowledge and domain specific knowledge which can be obtained from specialist sources by providing a general high-level structure in which to situate domain specific knowledge.

2. Agent Technology

Having reviewed the influences from knowledge engineering on our work, we will now look at those aspects of agent technology that we exploit. 'Software agency' is polysemous term. In this paper, it will be used to imply characteristics of flexibility, autonomy, intelligence, adaptability and high level communication in a piece of software. We hope that these characteristics will allow agents to give users an individually tailored perspective on information. This function will be indispensable in a world in which information is multiplying, but becoming less structured (Bradshaw 1996). In this role, agents could act as the medium through which people will access knowledge for new tasks from large networked repositories. Agents are already used on the Internet to filter and retrieve information in a number of applications. Bradshaw proposes a vision of the future in which,

> Ultimately, all data would reside in a "knowledge soup" where agents assemble and present small bits of information from a variety of data sources on the fly as appropriate to a given context.

In order to act in this way, the agent must be guided by some particular perspective or goal.

A popular Internet agent architecture is one in which the agent's behaviour is informed by some kind of user model (e.g. Lieberman 1995). For example, an information filtering agent may keep details of documents that it has already presented to the user, along with some indication of whether the user liked them or not. These records are used to produce a profile of the user's interests. This in turn guides the agent in its search for new documents to present to the user. This approach has some similarities to the IMPS approach. However, in IMPS, the model used to guide information retrieval is a task model rather than a user model. The profile maintained describes the requirements of a particular task type (selected by the user) in terms of domain information. Thus, an agent primed with a model of classification will be highly 'interested' in information about object classes and properties.

In addition to the characteristics mentioned above, agency can also indicate cooperative abilities; agents are often used not singly, but in a multi-agent architecture in which tasks are divided in some way between software entities. In this kind of architecture, agents can be distributed in space and time. Each agent can also hold separate logical and semantic beliefs. These properties can be exploited to cope with the scale and distributed, fast-changing and unreliable nature of the Internet. However, the advantages must be traded off against issues of problem dependent co-ordination (Bond and Gasser 1988).

In terms of the specific task of conducting knowledge acquisition on the Internet, one of the great advantages of the agent metaphor is that agents possess intentionality – the ability to express 'attitudes' towards information. This property becomes more significant in a multi-agent architecture where the agents express their intentionality in knowledge-level communication. An agent wanting to communicate fact X to another agent will do so in the form "I believe "All apples are red" to be true" or "I wish you to believe that "All apples are red"" rather than simply "All apples are red". This allows the second agent to reason about the fact that the first agent holds this belief, rather than simply having to accept "All apples are red" as a given. This is particularly important because information on the Web may be contradictory or simply incorrect. An intentional system has some capacity to reason about conflicting information.

3 The IMPS Architecture

Having outlined the theoretical roots of the IMPS architecture, this section will describe the architecture in specific terms. We will trace the ideas introduced earlier to their software implementations. IMPS is made up of agent programs. While these agents are independent, they cooperate at a high level to extract information from Internet sources. They reformulate this information so that it can be used in the kind of problem solving that is typically seen in knowledge based systems. To do this, the agents act together in a server architecture. The architecture will be described in two sections, the first detailing the internal structure and function common to all the IMPS agents and the second relating the agents together as a multi-agent system.

3.1 The Agent Level Architecture

Although each agent specializes in performing a certain task and may therefore have abilities that the other agents lack, all the agents are based on the same fundamental structure. This allows them to communicate with other IMPS agents via messages, retrieve information from the Web and manipulate it internally.

3.1.1 JAT, The Java Agent Template The basic structure on which all the IMPS agents are based is supplied by the Java Agent Template (JAT) 0.3 (Frost 1996). The JAT provides a template, written in the Java language, for constructing software agents that communicate peer-to-peer with a community of other agents distributed over the Internet. JAT agents are not migratory - in contrast to many other agent technologies, each IMPS agent has a static existence on a single machine. All agent messages use KQML as a top-level protocol or message wrapper (see section 3.1.3).

The template provides Java classes to support a virtual knowledge base for each agent, and includes functionality for dynamically exchanging "resources". These resources include Java classes such as 'languages' – these are essentially protocol handlers which enable a message to be parsed and provide some high level semantics. Another form of resource is an interpreter - a content handler, providing a procedural specification of how a message, constructed according to a specific ontology, should be interpreted. Data files can also be treated as resources.

The architecture of the JAT is ideal for prototyping and agent development. It was specifically designed to allow for the replacement and specialization of major functional components including the user interface, low-level messaging, message interpretation and resource handling.

3.1.2 Jess, The Java Expert System Shell The JAT agent template provides a range of common agent functions, such as the KQML messaging. In IMPS we have supplemented these functions with Jess which provides the 'brains' of each agent. Jess is a version of the popular expert system shell CLIPS, rewritten entirely in Java (Friedman-Hill 1998). It provides the agents with internal representation and inference mechanisms. In effect, the addition of Jess means that whilst the agents share a common architecture, each agent reasons and acts like a small knowledge-based system following its own set of rules. Jess can be used to manipulate external Java objects in a rule-based manner. This means the agent's low level behaviours can be directly controlled by the inference engine.

3.1.3 KQML, The Knowledge Query and Manipulation Language IMPS uses the Knowledge Query and Manipulation Language (KQML) for inter-agent communication, as specified and supported by the JAT. KQML has been proposed as a standard communication language for distributed applications in which agents communicate via "performatives" (Finin, Labrou and Mayfield 1997). It has a theoretical basis in speech-act theory. The sender explicitly represents and reasons about communication primitives and their effects in order to try to bring about specific mental states in the hearer (Jennings 1992). KQML is intended to be a high-level language to be used by knowledge-based systems to share knowledge rather than an internal representation language. As such, the semantics of its performatives refer specifically to agents' knowledge bases. Each agent appears to other agents to manage a knowledge base, whether or not this is true of the actual architectural implementation.

KQML supports the agent characteristic of intentionality, by allowing agents to communicate attitudes about information through performatives, such as querying, stating, believing, requiring, subscribing and offering. This provides the basis for an architecture in which agents have some kind of model of other agents, which in turn enables co-ordination. It also means that whatever internal knowledge representation mechanisms agents have, agent interaction is carried out at the knowledge level - it is implementation independent. KQML is indifferent to the format of the information itself, so expressions can contain sub-expressions in other languages. In IMPS, KIF statements are embedded in KQML. KIF is used for conveying the actual information content of knowledge bases, whilst KQML itself is used to convey the location of knowledge sources, agents and Java code modules.

3.1.4 KIF, The Knowledge Interchange Format Knowledge Interchange Format (KIF), a formal language for the interchange of knowledge among disparate computer programs, is used for content communication between agents. It has declarative semantics – so it is possible to understand the meaning of expressions in the language without appeal to an interpreter for manipulating those expressions. KIF has been designed to maximize translatability, readability and usability as a representation (Genesereth and Fikes 1992). Each IMPS agent has a KIF parser, written using the Java Compiler Compiler (JavaCC – see section 3.2.2), which allows it to read KIF text messages. KIF is maintained as a possible means of more sophisticated knowledge representation and sharing with other systems.

3.2 The Multi-Agent Architecture

As a model-driven architecture, IMPS aims to take the task-oriented nature of agent software much further. It uses PSM-oriented knowledge acquisition to create an explicit domain ontology for a task. The PSM used provides templates that describe the kind of knowledge required, the types of role that this knowledge might play and the inferences in which this knowledge might figure. The ontology provides a conceptual framework for the organization of knowledge. As it becomes instantiated with further structured acquisition, it produces a domain knowledge base that could in turn underpin agent-based problem solving guided by the same PSM structure.

3.2.1 The Server Architecture In order to apply these knowledge level models, IMPS uses a server architecture (see Figure 4), in which two specialist server agents, the Knowledge Extraction Agent (KExA) and the Ontology Construction Agent (OCA) provide knowledge to Inference Agents (IAs) on demand. The Inference Agents embody primitive inference types and co-ordinate to enact the PSM. This discussion and the prototype will focus on the server agents (see Section 4).

The KExA acts as a typical Agent Name Server (ANS) for a multi-agent system. It holds a registry of the names and locations of all active agents so that that this information can be served to other agents on request. Additionally, the KExA supervises the system's Knowledge Library (see Section 3.2.2), communicating to other agents the location and type of knowledge sources that are available to the system, and information about the problem solving method being used. It is also the interface between IMPS and the user during the first stages of system operation. It provides the simple knowledge acquisition interface through which the user enters information that is then translated into KIF so that all agents can store it in their knowledge bases.

The OCA uses the information communicated by the KExA to retrieve a PSM code module from the knowledge library. The PSM module contains information that will be used to structure a relevant domain ontology such as details of significant relationships. For example, heuristic classification requires a domain ontology that is strongly hierarchical, containing many 'is-a' relationships. The OCA will match the kind of relations required against the information sources that are available and the kind of relations they are likely to contain. It will then retrieve code modules from the library that will allow it to interface with the information sources selected. Usually, the OCA will begin by consulting a general online ontology to get a basic structure of terms around which it can build the domain ontology. It then assembles an appropriate external knowledge representation to store the information it extracts. Finally, it supplements the basic structure with information extracted from other networked sources.

The OCA has the ability to access and manipulate information from this representation and can reason over it. It uses this ability to integrate information from different sources and present an ontology graphically for negotiation with the user. Later, the OCA serves the information from the knowledge representation to the Inference Agents.

3.2.2 The Knowledge Library and Modularity The long-term aim of much of the work being done on knowledge sharing and reuse is that libraries of knowledge components such as domain and task ontologies be made available over the network. Therefore the ideal architecture for the future would seem to be one that is network-based, modular and extendible in such a way that it will be able to use new knowledge representation formats and standards as they arise.

The notion of modular PSMs is developed in IMPS - the PSMs themselves are not the smallest independent components. Smaller components - the KADS primitive inference types (Schreiber et al. 1998) - are embodied in agent shells to produce agents that specialize in performing a particular kind of inference (see Figure 2). For a classification task, agents might specialize in generation of classes, specification of attributes, or matching features. The knowledge based system arises from the dynamic configuration of problem solving agents reasoning over the external domain knowledge representation as served to them by the OCA.

When IMPS is used on the Internet, the PSM drives agent knowledge acquisition over highly implicit, heterogeneous and distributed knowledge sources. Therefore, standardization must occur at some point to allow the system to use uniform modular components. This happens in the knowledge library where the knowledge extraction modules and PSM modules are stored.

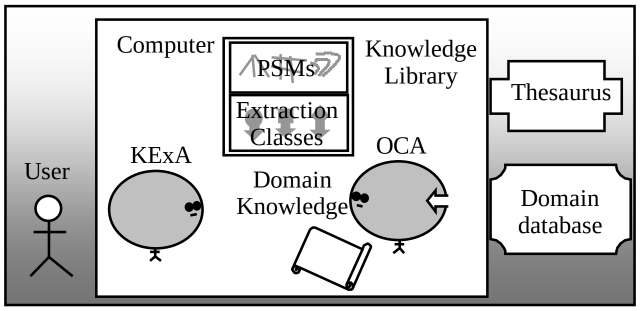


Figure 4: The IMPS server agents

The knowledge library component of IMPS is as essential to its operation as the agent component. The extraction classes used to obtain particular kinds of knowledge from knowledge sources are all based around a common Java interface, with standard inputs and outputs. The actual mechanisms by which the class extracts information from a source and parses it into a form comprehensible to Jess are completely hidden from the agent loading the class, according to the Object Oriented paradigm embodied in Java. New classes can be added to the library as appropriate, in a 'plug-and-play' manner, without any change to the rest of the architecture. This is also true of the PSM components, which are based around a (different) common interface. All the components in the library do not need to be held at the same physical location. They can be distributed across the network as long as they are registered with the Knowledge Extraction agent. Within the library, the knowledge sources are indexed by type - e.g. database, plain text file, etc., so new instances of a particular type of source just need to be identified as such to be used by the system.

Knowledge sources available to the IMPS architecture do not have to be static. In recognition of a global network in which an increasing amount of the information available is dynamically created, the classes can be written in such a way that they allow agents to interact with programs available over the network, such as search engines. In the prototype, the OCA uses a knowledge extraction class to interact with an interactive HTML search interface to the WordNet lexical database.

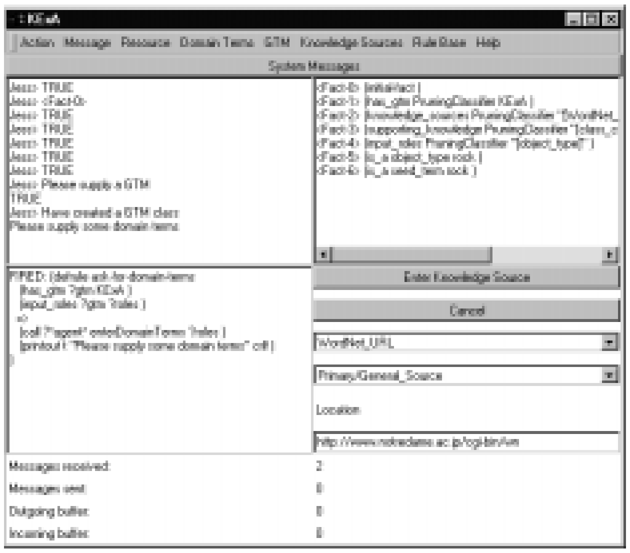
The open ended nature of the architecture anticipates the evolution of one or more meta-content languages on the Web (the frontrunner for this role seems to be extensible Markup Language (XML)). The widespread use of content-based tagging on the Web would increase the power of IMPS considerably by making the semantic structure of information presented on the Web more explicit. Once new standards are established, modules can be written for them using tools such as JavaCC - the Java compiler compiler. JavaCC is a tool that can automatically generate parsers if the grammar of a knowledge source can be expressed in a set format. If a grammar can be written for a new information format, then a code module can be generated semi-automatically to interpret it.

3.2.3 Knowledge Representation The principal domain knowledge representation is the domain ontology constructed and served to other agents by the OCA. Initially, the OCA uses a general online ontology in the form of a lexical database to get a basic structure, which is then supplemented with knowledge from domain specific sources. The general ontology used by IMPS is the WordNet semantically organized lexical database (Miller 1990) which contains approx. 57,000 noun word forms organized into around 48,800 word meanings (synsets). WordNet has advantages over other general ontologies in terms of a strong grounding in linguistic theory, on-line status and implemented search software (Beckwith and Miller 1990). In the long-term view, it has a Java interface suitable for agent interaction over the Internet, and several EuroWordNet projects are running.

Once a skeletal ontology has been created from a general ontology, the OCA supplements it with information obtained from data representations which contain explicit or implicit ontological statements, such as Ontolingua statements or relational database formats. This function serves as a 'proof of concept' for the design of a set of protocols (implemented as Java classes). These protocols would control the extraction of data from a variety of sources, such as databases, natural language text etc. that might be available to a distributed agent architecture e.g. over the Internet. A set of heuristic rules for the extraction of data from each kind of source could exploit the implicit ontologies inherent in the structure of information sources.

The domain ontology shared by the agents is used to facilitate problem solving communication relating to the domain. Problem solving knowledge is represented in the PSM modules stored in the IMPS knowledge library. These modules can be seen as indexed task ontologies describing each PSM in terms of input and output roles, inferences, and domain knowledge schema. These schema describe the way in which domain knowledge must be represented to be used by the PSM. As mentioned earlier, the PSM modules in IMPS are stored as Java classes using a common interface.

To present knowledge to the user, each agent has a user interface that is divided into panels displaying the state of the agent's inference engine (Figure 5). The top left panel shows the Jess output, used for communicating with the user. The bottom left panel shows the facts that the agent currently 'believes', and the bottom right panel shows the most recently fired rule, or the set of rules that are ready to fire. The top right panel is used for menu- and choice-based interactions with the user. The IMPS interface also shows the status of the agent as regards messaging (at the bottom) and has various menus at the top, depending on what kind of IMPS agent is being represented. The ontology agent has the ability to present the ontology it has created to the user in a graphical form.



```
Jess> TRUE
Jess> <Fact-0>
Jess> TRUE
Jess> TRUE
Jess> TRUE
Jess> TRUE
Jess> TRUE
Jess> Please supply a GTM
TRUE
Jess> Have created a GTM class
Please supply some domain terms
```

```
<Fact-0> (initial-fact )
<Fact-1> (has_gtm PruningClassifier KExA )
<Fact-2> (knowledge_sources PruningClassifier "(WordNet_
<Fact-3> (supporting_knowledge PruningClassifier "(class_c
<Fact-4> (input_roles PruningClassifier "(object_type)" )
<Fact-5> (is_a object_type rock )
<Fact-6> (is_a seed_term rock )
```

```
FIRED: (defrule ask-for-domain-terms
  (has_gtm ?gtm KExA )
  (input_roles ?gtm ?roles )
 =>
  (call ?*agent* ontoDomainTerms ?roles )
  (printout t "Please supply some domain terms" crlf )
)
```

WordNet_URL

Primary/General_Source

http://www.notredame.ac.jp/cgi-bin/wn

```
Messages received:   2
Messages sent:       0
Outgoing buffer:     0
Incoming buffer:     0
```

Figure 5: The KExA user interface

4 The Prototype

The prototype IMPS system (P-IMPS) focuses on the ontology construction stages of IMPS, rather than the later problem solving phase, constructing a domain ontology from online sources using two agents. The system has two PSMs – pruning classification and diagnosis, and two agents, which are:

- The Knowledge Extraction Agent (KExA), acting as an Agent Name Server (ANS) and the interface through which the user interacts with IMPS during initialization.
- The Ontology Construction Agent (OCA), which is able to use modules from the knowledge library to extract information from networked knowledge sources (in this example, WordNet, the online thesaurus/lexical database and a plain text domain database in the field of geology – the IGBA dataset).

Suppose a user is interested in performing a task in a domain but has little or no information to work with. Using a simple knowledge acquisition interface (Figure 5), the user provides the KExA with domain keywords, and chooses a PSM from a list.

The KExA selects and loads from the knowledge library a Java code module giving details of the PSM to be used. Then the user gives the names, types and URLs of some possible knowledge sources (specifying whether they are general sources, such as the thesaurus, or domain specific). The KExA passes on the PSM, domain and knowledge source information it has gathered to the OCA. The OCA then consults the knowledge library and extracts classes for handling the knowledge sources that have been specified.

Control rules organize the behaviour of the OCA into consecutive phases. The first phase is initialization – the OCA gets information from the PSM module about what kind of relationships between concepts and entities the PSM requires.

Next, the agent uses the general knowledge sources to get information on the domain keyword(s) given by the user – this involves matching between the kinds of information required by the PSM and the information available about the keyword(s) from the source. The latter is obtained through the knowledge source interface class which has a method for scanning the knowledge source and producing a list of the kinds of information that are likely to be available from it. This interface is implemented by all the knowledge extraction classes (in different ways). At this point, an interaction with the user occurs in which the OCA offers definitions of the keyword which have been extracted from the general source to the user for them to pick the one which most closely matches the concept they had in mind. A structured node in ontology representation is made for the concept represented by the keyword, containing all the information that has been found for it – synonyms, definitions etc. New nodes are also added for entities that are linked to the seed term in ways that are 'interesting' to the PSM. If the task structure is classification, the significant parts of the ontology will be hierarchical, and significant relations will be 'is-a' (to elicit object classes) and 'has-a' (to elicit attributes which will distinguish between the classes).

Next, the agent uses the generalized source to develop a simple ontology around the keyword concept. Each of the 'leaf' nodes that have been added is focused on in turn, and the generalized knowledge source is queried for new concepts related to these nodes in ways that are significant to the PSM. In order to keep memory and processor overheads at a minimum when creating a hierarchical ontology, depth first search is used to exhaust the possibilities for each path of related nodes before exploring a new one. Some extra control rules are used to filter out problems caused by inconsistency and repetition in the knowledge sources. When each node has been added to the external ontology representation, the OCA removes information relating to the nodes that have been created from its working memory, keeping the agent itself 'light' and fast.

The objects and relations extracted from the lexical database are presented back to the user graphically (Figure 6). This 'first pass' creates a skeletal ontology for the domain. The "HAS_PROPERTY" attributes of the nodes in Figure 6 have been generated by using very simple parsing on the textual definitions of concepts returned by WordNet.

Finally, the agent uses the secondary specialised source to supplement this ontology. To the agent, this phase is similar to the last one, but the sources used here are more specialised to the domain and are providing more detailed information. The information also needs to be integrated seamlessly into the existing ontology representation. The agent makes simple string-based matches between entities found in the new source and those already represented and confirms these with the user. These matches are used as points of reference in integrating new entities into the representation.

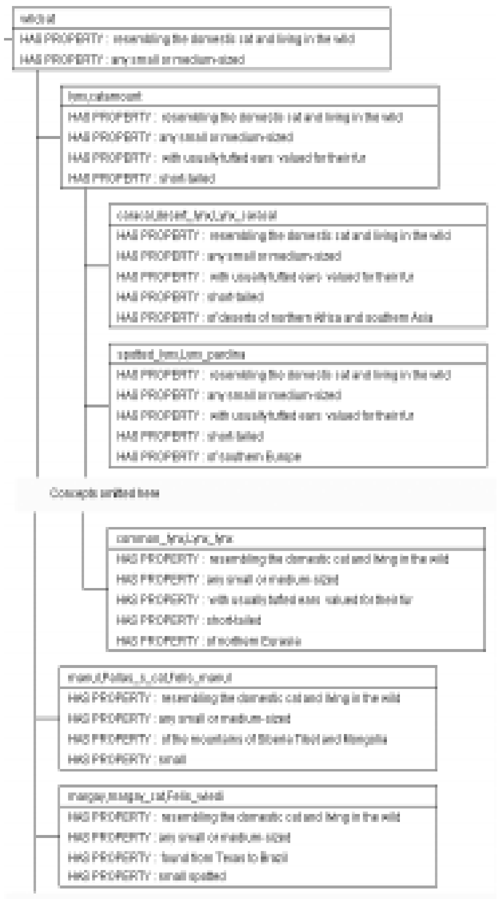

wildcat
HAS PROPERTY : resembling the domestic cat and living in the wild
HAS PROPERTY : any small or medium-sized
lynx,catamount
HAS PROPERTY : resembling the domestic cat and living in the wild
HAS PROPERTY : any small or medium-sized
HAS PROPERTY : with usually tufted ears valued for their fur
HAS PROPERTY : short-tailed
caracal,desert_lynx,Lynx_caracal
HAS PROPERTY : resembling the domestic cat and living in the wild
HAS PROPERTY : any small or medium-sized
HAS PROPERTY : with usually tufted ears valued for their fur
HAS PROPERTY : short-tailed
HAS PROPERTY : of deserts of northern Africa and southern Asia
spotted_lynx,Lynx_pardina
HAS PROPERTY : resembling the domestic cat and living in the wild
HAS PROPERTY : any small or medium-sized
HAS PROPERTY : with usually tufted ears valued for their fur
HAS PROPERTY : short-tailed
HAS PROPERTY : of southern Europe
Concepts omitted here
common_lynx,Lynx_lynx
HAS PROPERTY : resembling the domestic cat and living in the wild
HAS PROPERTY : any small or medium-sized
HAS PROPERTY : with usually tufted ears valued for their fur
HAS PROPERTY : short-tailed
HAS PROPERTY : of northern Eurasia
manul,Pallas_s_cat,Felis_manul
HAS PROPERTY : resembling the domestic cat and living in the wild
HAS PROPERTY : any small or medium-sized
HAS PROPERTY : of the mountains of Siberia Tibet and Mongolia
HAS PROPERTY : small
margay,margay_cat,Felis_wiedi
HAS PROPERTY : resembling the domestic cat and living in the wild
HAS PROPERTY : any small or medium-sized
HAS PROPERTY : found from Texas to Brazil
HAS PROPERTY : small spotted

Figure 6: Fragment of an ontology of cats developed for a classification PSM

Different heuristic methods are used for extraction. For example, the extraction module used with databases identifies 'unique' columns in the database in which each row has a different value, and 'classifications', in which each value may appear more than once. It is inferred that the unique values may be hyponyms (sub-classes) of the classifications. They are then added to the ontology. Matches between existing ontology entities and concepts found in the database are used to position them correctly. This process creates an enriched ontology in terms of both size and correct domain representation. In the domain of geology, using only this simple heuristic rule, around 200 new entities were added to an ontology of igneous rocks at this stage. The categorical concepts ‘volcanic’ and ‘plutonic’ were matched with existing concepts in the ontology and the new subclass entities were added in hyponymical relationships to the concepts (Figure 7). The concepts marked with numbers have been added from the second source.

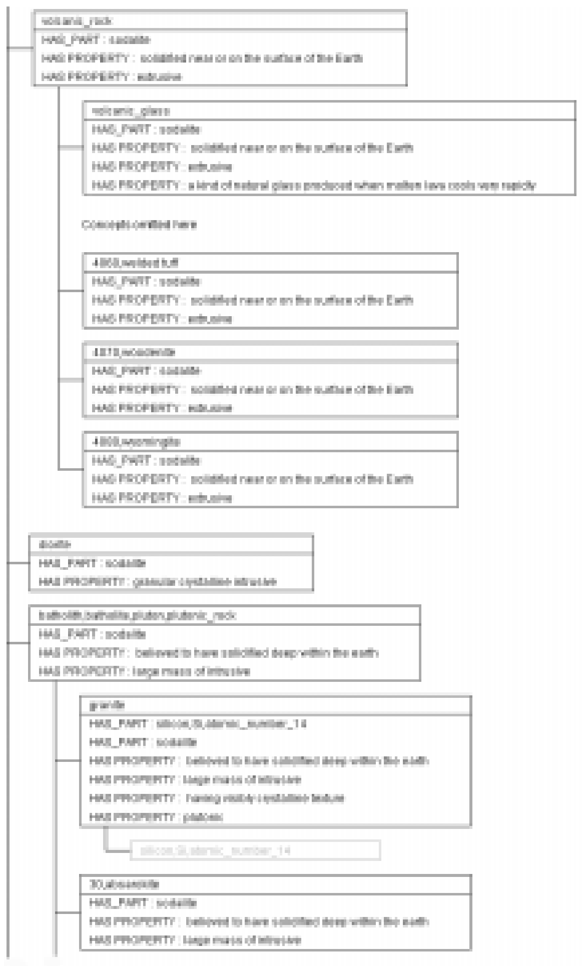

volcanic_rock
HAS_PART : sodalite
HAS PROPERTY : solidified near or on the surface of the Earth
HAS PROPERTY : extrusive
volcanic_glass
HAS_PART : sodalite
HAS PROPERTY : solidified near or on the surface of the Earth
HAS PROPERTY : extrusive
HAS PROPERTY : a kind of natural glass produced when molten lava cools very rapidly
Concepts omitted here
4868,welded tuff
HAS_PART : sodalite
HAS PROPERTY : solidified near or on the surface of the Earth
HAS PROPERTY : extrusive
HAS_PART : sodalite
HAS PROPERTY : solidified near or on the surface of the Earth
HAS PROPERTY : extrusive
4889,wyomingite
HAS_PART : sodalite
HAS PROPERTY : solidified near or on the surface of the Earth
HAS PROPERTY : extrusive
diorite
HAS_PART : sodalite
HAS PROPERTY : granular crystalline structure
batholith,batholite,pluton,plutonic_rock
HAS_PART : sodalite
HAS PROPERTY : believed to have solidified deep within the earth
HAS PROPERTY : large mass of intrusive
granite
HAS_PART : silicon,Si,atomic_number_14
HAS_PART : sodalite
HAS PROPERTY : believed to have solidified deep within the earth
HAS PROPERTY : large mass of intrusive
HAS PROPERTY : having visibly crystalline texture
HAS PROPERTY : plutonic
silicon,Si,atomic_number_14
30,absarokite
HAS_PART : sodalite
HAS PROPERTY : believed to have solidified deep within the earth
HAS PROPERTY : large mass of intrusive

Figure 7: Fragment of an ontology of igneous rocks developed for a classificatory PSM

As the process continues, it becomes clear that there is a thin line between this kind of ontology construction and knowledge acquisition. The same model could be used to describe the acquisition of domain knowledge from multiple sources. The real power is the amount of automated domain acquisition that has been achieved by using software agents driven by knowledge level models.

It should be noted that there is a wide variety of PSMs in standard libraries. If a different PSM is used, the concepts and relationships featured in the ontology are qualitatively different. For example, if the prototype is initialized with a diagnosis PSM and a domain term from the field of medicine, the initial ontology produced is structurally very different, as the focus is on causal relationships (see Figure 8). Again, the causal attributes are extracted by simple textual parsing.

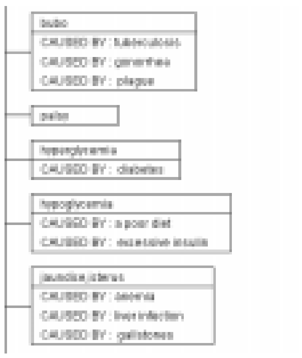


Figure 8: Fragment of an ontology of symptoms developed for a diagnosis PSM

5 Evaluation

It is difficult to thoroughly evaluate a system such as the IMPS prototype, which draws on several areas of research and integrates various technological components. However, some aspects of the system can be earmarked as suitable for formal evaluation. The ontologies created by IMPS are the most obvious example. We plan to perform empirical evaluations on the ontologies produced by the prototype to test several hypotheses.

1. The IMPS system produces initial domain ontologies for tasks that are comparable in quality to those produced by hand by domain novices experienced in knowledge acquisition. The novices and IMPS will use the same knowledge representation and the same knowledge sources.
2. The IMPS system can use PSM oriented domain schema to produce initial domain ontologies that are adequate for problem solving. These ontologies are superior in terms of problem solving to those produced by IMPS using another PSM domain schema or those produced by IMPS without using a PSM domain schema.
3. IMPS produces ontologies using two knowledge sources that are superior to those produced using a single source.
4. The ontologies produced by IMPS using a single source are superior to the ontologies represented implicitly in the source itself.
5. IMPS can construct meaningful ontologies using the same PSM oriented domain schema in different domains.

All the hypotheses can be tested using an experimental design in which ontologies will be presented to domain experts who will evaluate them. This evaluation will be based on qualitative measures of ontology 'goodness' as used in Tennison's (Tennison 1998) evaluation of a collaborative ontology construction tool. The criteria used in that evaluation were precision, breadth, consistency, completeness, readability and utility.

The experimental design for hypothesis 1 will involve a comparison between the IMPS ontologies and handcrafted ontologies created by subjects who are experienced both in knowledge engineering (i.e. in the process of eliciting and representing knowledge) and in using the Web. These subjects will not, however, be expert in the domains in which they will be constructing ontologies.

The reasoning behind the choice of subjects in this experiment is that these KA-experienced subjects are the kind of people who might be expected to use a system such as the IMPS prototype to assist them. Therefore, if IMPS can show significant improvements over what the subjects can do by hand, this will be an indication that it could be an asset in the knowledge acquisition process.

As mentioned already, we intend to develop the IMPS system further so that it can use documents that have been marked up in an XML-compliant way as secondary knowledge sources. This will allow the system to operate fully over a much wider range of domains. This expansion of scope will have the secondary effect of making evaluation an easier task in terms of finding experts to evaluate the ontologies (see comments on rarity of experts in section 1).

Some aspects of the system are not yet mature enough for formal evaluation. This is the case with the Inference Agents that embody inference steps from PSMs and act over the domain ontologies. Long-term evaluation aims would be to assess whether the complete IMPS architecture could perform PSM construction and opportunistic collaboration, together with ontology construction and knowledge, to create an on-line expert system on-the-fly.

6 Conclusions

In this paper, we have described a multi-agent approach to the problem of getting useful knowledge from distributed and implicit information sources. In (Crow and Shadbolt 1998) we describe the position of IMPS with respect to other related work. The IMPS architecture features clean syntactic integration of information presented originally in different formats, and rule-based semantic information integration. It does this through the use of standardised modular components to filter information while maintaining a lightweight agent architecture. The modular design means that IMPS is open to the use of new technologies and standards.

IMPS applies knowledge level models to Web information sources through a server architecture. In this way, it accomplishes preliminary ontology construction, KA and problem solving. The architecture does not merely represent a 'bolting together' of existing technologies. It proposes a solution to the research issue of what components are necessary to create a rational and scalable architecture. It demonstrates that significant knowledge acquisition and problem solving agents can be designed and implemented, and that complex communication between them can be generated and implemented in the light of domain ontologies, problem solving models and acquisition activities.

Acknowledgements

Louise Crow is supported by a University of Nottingham Research Scholarship.

References

Bateman, J.; Magnini, B.; and Fabris, G. 1995. The generalized upper model knowledge base: Organization and use. In Mars, N. ed., *Towards very large knowledge bases: knowledge building and knowledge sharing* 60-72. Amsterdam.:IOS Press.

Beckwith, R., and Miller, G. A. 1990. Implementing a lexical network. *International Journal of Lexicography* 3 (4): 302 - 312.

Benjamins, R. 1997. Problem-Solving Methods in Cyberspace. In Proceedings of the Workshop on Problem-Solving Methods for Knowledge-based Systems at the Fifteenth International Joint Conference on Artificial Intelligence. Nagoya, Japan.: International Joint Conferences on Artificial Intelligence, Inc.

Bond, A. H., and Gasser, L. eds. 1988. *Readings in Distributed Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann.

Bradshaw, J. ed. 1996. *Software Agents.* Menlo Park, CA: AAAI Press/The MIT Press.

Bylander, T., and Chandrasekaran, B. 1988. Generic tasks in knowledge-based reasoning: the right level of abstraction for knowledge acquisition. In Gaines, B and Boose, J. eds. *Knowledge Acquisition for Knowledge-based Systems* 1:65-77. London: Academic Press.

Chandrasekaran, B. 1986. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert* 1 (3): 23-30.

Crow, L. R., and Shadbolt, N. R. 1998. Internet Agents for Knowledge Engineering. In Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Canada.: SRDG Publications.

Cycorp, Inc. 1997. The Cycorp homepage. WWW: http://www.cyc.com/

Fensel, D. 1997. An Ontology-based Broker: Making Problem-Solving Method Reuse Work. In Proceedings of the Workshop on Problem-Solving Methods for Knowledge-based Systems at the Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan.: International Joint Conferences on Artificial Intelligence, Inc.

Finin, T.; Labrou, Y.; and Mayfield, J. 1997. KQML as an agent communication language. In Bradshaw J. M. ed. *Software Agents*. Cambridge, MA.: AAAI/MIT Press.

Friedman-Hill, E. J. 1998. Jess, The Java Expert System Shell, Technical Report, SAND98-8206 (revised), Sandia National Laboratories, Livermore. WWW: http://herzberg.ca.sandia.gov/jess

Frost, H. R. 1996. Documentation for the Java(tm) Agent Template, Version 0.3. Center for Design Research, Stanford University. WWW: http://cdr.stanford.edu/ABE/documentation/index.html

Genesereth, M. R., and Fikes, R. E. 1992. Knowledge Interchange Format Version 3.0 Reference Manual, Technical Report, Logic-92-1, Computer Science Department, Stanford University.

Gil, Y., and Melz, E. 1996. Explicit representations of problem-solving strategies to support knowledge acquisition. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 469-476. Menlo Park, CA.: AAAI Press/MIT Press.

Gruber, T. R. 1993. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition* 5(2): 199-220.

Jennings, N. R. 1992. Towards a cooperation knowledge level for collaborative problem solving. In Proceedings of the Tenth European Conference on Artificial Intelligence, 224-228. Vienna, Austria.: John Wiley & Sons.

Klinker, G.; Bhola, C.; Dallemagne, G.; Marques, D.; and McDermott, J. 1991. Usable and reusable programming constructs*. Knowledge Acquisition* 3 (2): 117-136.

Lawrence, S., and Giles, C. L. 1998. Searching the World Wide Web. *Science* 280: 98-100.

Lieberman, H. 1995. Letizia: An Agent that Assists Web Browsing. In Working Notes of AAAI-95 Fall Symposium Series on AI Applications in Knowledge Navigation and Retrieval 97-102. Cambridge, MA.: The AAAI Press.

Miller, G. 1990. WordNet: An on-line lexical database. International Journal of Lexicography, Vol. 3 (4): 235-312.

Newell, A. 1982. The Knowledge Level. *Artificial Intelligence* 18: 87-127.

Puerta, A. R.; Eriksson, H.; Egar, J. W.; and Musen, M. A. 1992. Generation of Knowledge-Acquisition Tools from Reusable Domain Ontologies, Technical Report KSL 92-81 Knowledge Systems Laboratory, Stanford University.

Schreiber, A Th.; Akkermans, J. M.; Anjewierden A. A.; de Hoog H., Shadbolt, N. R.; Van de Velde, W.; and Weilinga, B. J. 1998. Engineering and Managing Knowledge. The CommonKADS Methodology [version 1.0]. Amsterdam, The Netherlands.: Department of Social Science Informatics, University of Amsterdam.

Steels, L. 1990. Components of Expertise. *AI Magazine*, Vol. 11 (2):29-49.

Studer, R.; Eriksson, H.; Gennari, J.; Tu, S.; Fensel, D.; and Musen, M. 1996. Ontologies and the Configuration of Problem-solving Methods. In Proceedings of the Tenth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Canada.: SRDG Publications.

Swartout, B.; Patil, R.; Knight, K.; & Russ, T. 1996. Toward Distributed Use of Large-Scale Ontologies. In Proceedings of the Tenth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Canada.: SRDG Publications.

Tennison, J. 1998. Collaborative Knowledge Environments on the Internet. Forthcoming Ph.D. Thesis, Dept of Psychology, University of Nottingham.

Tennison, J., and Shadbolt, N. R. 1998. APECKS: A Tool to Support Living Ontologies. In Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Canada.: SRDG Publications.

Uschold, M. 1998. Knowledge Level Modelling: Concepts and terminology. *The Knowledge Engineering Review* 13 (1): 5-30.

Van de Velde, W. 1993. Issues in Knowledge Level Modelling. In David, J. M., and Krivine, J. P, and Simmons, R. eds. *Second Generation Expert Systems.* Berlin.: Springer Verlag.

van Heijst, G.; Schreiber, A. Th.; and Wielinga, B. J. 1997. Using Explicit Ontologies for KBS Development. *International Journal of Human-Computer Studies/Knowledge Acquisition*, 2(3):183-292.

Walther, E.; Eriksson, H.; and Musen, M. 1992. Plug-and-Play: Construction of task-specific expert-system shells using sharable context ontologies. Technical Report KSI-92-40, Knowledge Systems Laboratory, Stanford University.

Weilinga, B.; Van de Velde, W.; Schreiber, G.; and Akkermans, H. 1991. Towards a unification of knowledge modelling approaches, Technical Report KADS-II/T1.1/UvA/004/2.0, Dept. of Social Science Informatics, University of Amsterdam.