The Cycic Friends Network: getting Cyc agents to reason together[1]

James Mayfield, Tim Finin,
Rajkumar Narayanaswamy and Chetan Shah
University of Maryland Baltimore County
Baltimore MD

William MacCartney and Keith Goolsbey
Cycorp, Austin TX

December 1995

Abstract

We describe the initial results of a project aimed at adapting the Cyc system for use in an agent architecture. Two Cyc systems that share a large common core of knowledge but differ in additional knowledge they possess were able to reason together to solve problems that neither could solve on its own. A rudimentary interface was constructed for Cyc that allowed it to communicate with other KQML-speaking agents. The Cyc reasoning algorithm was modified to allow it to ask other agents for help in developing a proof to answer a query. We were able to demonstrate that Cyc can be adapted to work in an agent-oriented architecture and to get several Cyc-based agents to reason in a tightly-coupled manner. A number of interesting research issues remain concerning how to do so efficiently.

---

[1] This work was supported in part by the Department of Defense under contract MDA904-95-C-2084, by the Air Force Office of Scientific Research under contract F49620-92-J-0174, and by the Advanced Research Projects Agency monitored under USAF contracts F30602-93-C-0177 and F30602-93-C-0028 by Rome Laboratory.

1. Introduction

A key feature of the agent model of computation is that agents (be they personal agents or distributed agents) must have access to various kinds of expertise. Each agent will embody some, but perhaps not all, of the expertise it will need to conduct its affairs. Two agents might be expected to share a large core of knowledge, but no two agents are likely to hold exactly the same knowledge. Thus, an agent might be justified in seeking help from another agent to perform inference in an area for which it lacks expertise.

The distribution of inference across several agents has been studied for years under the rubric of "Distributed AI." However, traditional DAI approaches are unlikely to work well in an agent architecture for use across the Internet, both because shared resources (such as a blackboard) will not be readily available, and because communication costs will often dominate computation costs.

The purpose of our system, the Cycic Friends Network (CFN), is to explore cooperative inference across the Internet. We are using the Cyc knowledge base and inference engine as our testbed. We are building a system that distributes inference across two or more Cyc-based agents. We aim to develop a system architecture that will allow for dynamic resource discovery, that will minimize communication costs, and that will be upwardly scaleable.

Our approach to inference in this environment is to place overall control of the inference process in the hands of a single initiating agent. The KQML language will allow the controlling agent to send unsolved subgoals to other agents. We adopt a mediated architecture to allow an agent to locate other agents with the desired expertise.

In the following sections, we first discuss Cyc and KQML. We then describe how we adapted the Cyc inference engine to handle distributed inference. After presenting two examples, we show how this inference mechanism has been embedded in an agent architecture. Finally, we discuss open research issues, and give our conclusions.

2. The Cyc System

2.1 The Philosophy of the Cyc Project

Cyc is a large, multi-contextual knowledge base and inference engine developed by Cycorp, Inc., in Austin, Texas. The goal of the Cyc project is to break the "software brittleness bottleneck" by constructing a foundation of basic commonsense knowledge—a semantic substratum of terms, rules, and relations—that will enable a variety of knowledge-intensive products and services. Cyc is intended to provide a "deep" layer of understanding that can be used by other programs (such as domain-specific expert systems) to make them more flexible. Cyc has provided the foundation for ground-breaking pilot applications in the areas of heterogeneous database browsing and integration, captioned image retrieval, and natural language processing.

2.2 The Cyc Knowledge Base

The Cyc knowledge base (KB) is a formalized representation of a vast quantity of fundamental human knowledge: facts, rules of thumb, and heuristics for reasoning about the objects and events of everyday life. The medium of representation is the formal language CycL, described below. The KB consists of terms—which constitute the vocabulary of CycL—and assertions that relate those terms. These assertions include both simple ground assertions and rules. Cyc is not a frame-based system: the Cyc team thinks of the KB instead as a sea of assertions, with each assertion being no more "about" one of the terms involved than another.

The Cyc KB is divided into about 100 "microtheories," each of which is a bundle of assertions focused on a particular domain of knowledge. The microtheory mechanism allows Cyc to independently maintain assertions that are *prima facie* contradictory, and enhances the performance of the Cyc system by focusing the inference process.

At this writing, the Cyc KB contains approximately 50,000 terms and approximately 500,000 assertions. New assertions are continually added to the KB both by human knowledge enterers and by Cyc itself as a product of the inference process.

2.3 CycL: The Cyc Representation Language

CycL, the Cyc representation language, is an extraordinarily flexible knowledge representation language that augments first-order predicate calculus (FOPC) with extensions to handle equality, default reasoning, skolemization, and some second-order features. (For example, quantification over predicates is allowed in some circumstances, and complete assertions can appear as intensional components of other assertions.) CycL uses a form of circumscription, includes the unique names assumption, and can use the closed world assumption where appropriate.

2.4 Inference in Cyc

The Cyc inference engine performs general logical deduction (including *modus ponens, modus tolens,* and universal and existential quantification), using various well-known named inference mechanisms (inheritance, automatic classification, *etc.*) as special cases. Cyc performs best-first search over proof-space using a set of proprietary heuristics, and uses microtheories to optimize inference by restricting search domains.

The Cyc KB contains hundreds of thousands of assertions. Many approaches commonly taken by other inference engines (such as frames, RETE match, Prolog, *etc.*) do not scale well to KBs of this size. As a result, the Cyc team has been forced to develop new techniques.

Cyc also includes several special-purpose inference modules for handling a few specific classes of inference. One such module handles reasoning concerning collection membership and disjointness. Others handle equality reasoning, temporal reasoning, and mathematical reasoning.

3. Knowledge Query and Manipulation Language

To address many of the difficulties of communication among intelligent agents, we must give them a common language. In linguistic terms, this means that they must share a common syntax, semantics and pragmatics. Getting information processes or software agents to share a common syntax is a major problem. There is no universally accepted language in which to represent information and queries. Languages such as KIF, extended SQL, and LOOM have their supporters, but there is also a strong position that it is too early to standardize on any representation language. As a result, it is currently necessary to say that two agents can communicate with each other if they have a common representation language or use languages that are inter-translatable.

Assuming the use of a common or translatable language, it is still necessary for communicating agents to share a framework of knowledge to interpret the messages they exchange. This is not really a shared semantics, but rather a shared ontology. There is not likely to be one shared ontology, but many. Shared ontologies are under development in many important application domains such as planning and scheduling, biology and medicine. The pragmatics of communicating software agents involves such issues as knowing with whom to talk and how to find them as well as knowing how to initiate and maintain an exchange. The Knowledge Query and Manipulation Language (KQML) is concerned primarily with this kind of pragmatics and secondarily with semantics. It is a language and a set of protocols that support software agents in identifying, connecting with and exchanging information with other agents. In this section we present the KQML language, its primitives and supported protocols, and the software environment of KQML-speaking applications.

The KQML language consists of three layers: the content layer, the message layer, and the communication layer. The content layer bears the actual content of the message, in the agents' own representation language. KQML can carry expressions encoded in any representation language, including languages expressed as ASCII strings and those expressed using a binary notation. Some KQML-speaking agents (*e.g.*, routers, general brokers, *etc.*) may ignore the content portion of the message, except to determine where it ends.

The communication level encodes a set of message features that describe the lower level communication parameters, such as the identity of the sender and recipient, and a unique identifier associated with the communication.

The message layer is used to encode a message that one application would like to transmit to another. The message layer forms the core of the KQML language, and determines the kinds of interactions one can have with a KQML-speaking agent. A primary function of the message layer is to identify the protocol to be used to deliver the message, and to supply a speech act or performative that the sender attaches to the content (such as that it is an assertion, a query, a command, or any of a set of known performatives). In addition, since the content may be opaque to a KQML-speaking agent, this layer also includes optional features that describe the content language, the ontology it assumes, and a description of the content (such as a descriptor naming a topic within the ontology). These features make it possible for KQML implementations to analyze, route and properly deliver messages despite the inaccessibility of their content.

The syntax of KQML is based on a balanced parenthesis list. The initial element of the list is the performative; the remaining elements are the performative’s arguments as keyword/value pairs. Because the language is relatively simple, the actual syntax is not significant and can be changed if necessary in the future. The syntax reveals the roots of the early implementations, which were done in Common Lisp; it has proven to be quite flexible.

A KQML message from agent *joe* representing a query about the price of a share of IBM stock might be encoded as:

```
(ask-one
   :sender joe
   :content (PRICE IBM ?price)
   :receiver stock-server
   :reply-with ibm-stock
   :language LPROLOG
```

```
:ontology NYSE-TICKS)
```

In this message, the KQML performative is ask-one, the content is (PRICE IBM ?price), the ontology assumed by the query is identified by the token NYSE-TICKS, the receiver of the message is to be a server identified as *stock-server* and the query is written in a language called *LPROLOG*.

The value of the :content keyword forms the content level; the values of the :reply-with, :sender, and :receiver keywords form the communication layer; and the performative name (ask-one), together with the :language and :ontology keywords form the message layer.

In due time, *stock-server* might send to *joe* the following KQML message:

```
(tell
    :sender stock-server
    :content (PRICE IBM 96.625)
    :receiver joe
    :in-reply-to ibm-stock
    :language LPROLOG
    :ontology NYSE-TICKS)
```

A query similar to the above ask-one query could be conveyed using standard Prolog as the content language in a form that requests the set of all answers as:

```
(ask-all
    :content "price(ibm, Price)"
    :receiver stock-server
    :language standard_prolog
    :ontology NYSE-TICKS)
```

The original message asks for a single reply; this second request message asks for a set of answers as a reply. If we prefer each response to be sent separately instead of as a single large collection, we can use the stream-all performative (to save space, we will no longer repeat fields that are the same as in the above examples):

```
(stream-all
    :comment "?VL is a large set of symbols"
    :content (PRICE ?VL ?price))
```

The stream-all performative asks that a set of answers be turned into a stream of replies. To exert control over this set of reply messages we can wrap another performative around the preceding message:

```
(standby
    :content (stream-all
    :content (PRICE ?VL ?price)))
```

The standby performative expects a KQML expression as its content. It requests that the agent receiving the request receive the stream of messages one at a time; each time, the sending agent transmits a message with the next performative. The exchange of next/reply messages can continue until the stream is depleted or until the requesting agent sends a discard message (*i.e.* discard all remaining replies) or a rest message (*i.e.* send all of the remaining replies now).

A different set of answers to the same query can be obtained (from a suitable server) with the query:

```
(subscribe
    :content (stream-all
                :content (PRICE IBM ?price)))
```

This performative requests all future changes to the answer to the query, *i.e.*, it requests that a stream of messages be generated to reflect changes in the trading price of IBM stock.

Although KQML defines a set of reserved performatives, it is neither a minimal required set nor a closed one. A KQML agent may choose to handle only a few (perhaps one or two) performatives. The set is extensible; a community of agents may choose to use additional performatives if they agree on their interpretation and the protocol associated with each. However, an implementation that chooses to implement one of the reserved performatives must implement it in the standard way.

Some of the reserved performatives are shown in Figure 1. In addition to standard communication performatives such as <u>ask</u>, <u>tell</u>, <u>deny</u>, <u>delete</u>, and protocol-oriented performatives such as <u>subscribe</u>, KQML contains performatives related to the non-protocol aspects of pragmatics, such as <u>advertise</u> (which allows an agent to announce what kinds of KQML messages it is willing to handle); and <u>recruit</u> (which can be used to find suitable agents for particular types of messages). For example, the server in the above example might have earlier announced:

```
(advertise
    :ontology NYSE-TICKS
    :language LPROLOG
    :content (stream-all
                :content (PRICE ?x ?y)))
```

This is roughly equivalent to announcing that it is a stock ticker and inviting monitor requests concerning stock prices. This *advertise* message is what justifies the subscriber's sending the *stream-all* message.

Category	Name
Basic query	evaluate, ask-if, ask-about, ask-one, ask-all
Multi-response query	stream-about, stream-all, eos
Response	reply, sorry
Generic informational	tell, achieve, cancel, untell, unachieve
Generator	standby, ready, next, rest, discard, generator
Capability-definition	advertise, subscribe, monitor, import, export
Networking	register, unregister, forward, broadcast, route

Figure 1. KQML has about two dozen reserved performative names, which fall into these seven categories.

4. Adapting Cyc to an Agent Architecture

Inference in Cyc can be viewed as a search through proof-space. Each node in the search tree contains the following information:

parent:	the parent node of this node in the search
formula:	the current set of literals that we are attempting to prove
via:	the assertion used to infer our current formula from our parent's formula
mt:	the current microtheory in which we are doing inference

Cyc performs best-first search over this search space using a set of proprietary heuristics. After choosing the most promising outstanding unexpanded node, the node is examined to determine how it might be expanded.

The standard expansion options are:

lookup:	simplify using ground assertions from the KB
backchain:	backchain using rules from the KB

These options use formulas from the knowledge base to unify and transform the current formula using standard resolution theorem proving. Cyc provides a way to extend the expansion options beyond these standard ones through the addition of heuristic-level modules (HL modules). These modules are special-purpose reasoning tools designed to efficiently handle formulas with particular syntactic forms. Each HL module has three parts:

enabler:	is this HL module currently available?
recognizer:	when given a search node, does the HL module apply?
expander:	produces all child nodes of an applicable search node

Among the implemented HL modules are:

isa:	handle formulas involving #$instanceOf
genls:	handle formulas involving #$genls (*i.e.,* superset)
equality:	handle formulas involving #$equals

To allow inference to span two agents in CFN, we added a new HL module:

external:	handle any formula by asking a different Cyc agent

The enabler for this HL module is merely a control variable. Its simplest recognizer is trivial—all formulas pass. The expander performs a remote function call to the other Cyc agent asking it to expand the search node as if it were one of its own. The process works like this:

1. Agent A (the caller) decides to use its external HL module to ask Agent B (the callee) to expand the current search node.

2. Agent A passes the state information from the current search node to Agent B.

3. Agent B creates a single search node from this information.

4. Agent B then expands this single node just as if it were a node in one of its own inferences. This produces a (possibly empty) set of child nodes. To prevent possible infinite looping, Agent B disables its external HL module during expansion; this places the inference process squarely under the control of the single initiating agent.

5. Agent B returns to Agent A a list containing the state information for each of the child nodes that it computed.

6. Agent A creates a child node from each of the items it receives from Agent B.

If Agent B has knowledge that Agent A does not, in step (4) it will be able to expand the node in ways that Agent A could not. Agent A is then able to continue reasoning even though it was unable to perform one of the steps itself—it effectively trusts the answer from Agent B and forges on.

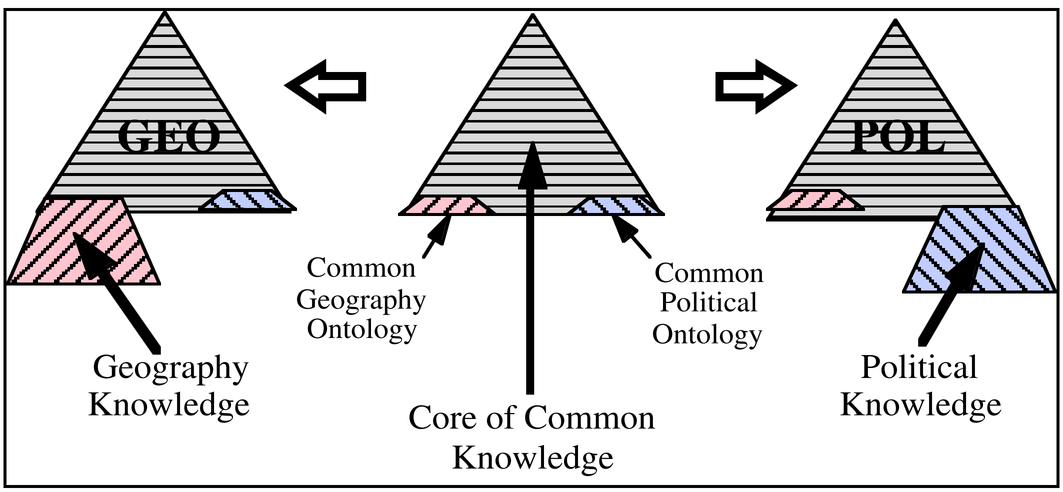

Figure 2. The core of common knowledge and largely disjoint extensions minimize chances of conflicts, contradictions and inconsistencies.

5. The GeoPolitical Example

Consider the following two queries:

- Which Middle Eastern countries are believed likely to attack another country?
- Who are some elected heads of state of countries north of the equator?

Answering each of these requires drawing on knowledge from two different domains: geography and politics. The second query is expressed in CycL as:

```
;; elected heads of government of countries north of the equator
(#$LogAnd
  (#$headOfGovernmentOf $x $y)
  (#$hasAttributes $x #$Elected)
  (#$northOf $y #$Equator))
```

To demonstrate the ability of the multiple-Cyc-agent architecture to perform distributed inference, we created two distinct Cyc agents running on separate machines that possessed slightly different knowledge bases. This organization is shown in Figure 2. Both agents held the very large set of core knowledge common to all Cyc agents. Moreover, both agents possessed the vocabulary for talking about both the geography domain and the politics domain. But one of the agents (designated the Geo agent) possessed, in addition, specific facts and rules about the geography domain, while the other (the Pol agent) possessed specific facts and rules about the politics domain.

Common Knowledge

The Core Cyc KB (~500,000 assertions).

Vocabulary for both the geography domain and the politics domain (bordersOn, northOf, Equator, headOfStateOf, ElectedLeader, *etc.*)

The Geo Agent

All of the above, plus:

Ground assertions stating:

- which countries border on which other countries;
- which countries are in which regions, continents, and hemispheres;
- which continents are in which hemispheres; and
- the northern hemisphere is north of the equator.

A rule stating:

- If region A is part of region B, and region B is direction D from region C, then region A is direction D from region C.

The Pol Agent

All of the above, plus:

Ground assertions stating:

- the types of governments (democratic, communist, authoritarian, *etc.*) of several countries;
- the level of military strength of several countries; and
- the heads of state of several countries.

Rules stating:

- If a country with an authoritarian government and high military strength borders on a country with low military strength, then the former is likely to attack the latter.
- If a country has a democratic government, then the head of state of that country is an elected leader.

If all the knowledge described above were present in one Cyc agent, that agent could easily give answers to the example queries, using chains of inference such as the following to answer our two example queries. For the first query, *"Which Middle Eastern countries are believed likely to attack another country?,"* the following reasoning leads to the answer *"Iraq is likely to attack Kuwait."*

- If a country with an authoritarian government and high military strength borders on a country with low military strength, then the former is likely to attack the latter.
- Iraq has an authoritarian government.
- Iraq has a strong military.
- Kuwait has a weak military.
- Iraq borders on Kuwait.
- Iraq is located in the Middle East.

The second example, *"Who are some elected heads of state of countries north of the equator?"* results in the answer *"John Major"* by the following reasoning:

- If a country has a democratic government, then the head of state of that country is an elected leader.
- Great Britain has a democratic government.
- John Major is the head of state of Great Britain.
- If region A is part of region B, and region B is north of region C, then region A is north of region C.
- Europe is in the northern hemisphere.
- The northern hemisphere is north of the equator.
- Great Britain is in Europe.

Note that both example queries depend on knowledge from both domains. Example One draws on ground assertions from the geography domain, and on both ground assertions and a rule from the politics domain. Example Two goes further, drawing on both ground assertions and a rule from each of the two domains. In addition, Example Two requires a repeated application of a rule in the geography domain.

6. **Agent Architecture**

The architecture we use for distributed problem solving and cooperative information processing uses the *brokerage* concept, wherein a centralized agent termed the 'broker' provides mediation and translation services. Effective matchmaking between servers and clients is provided by a knowledge-sharing infrastructure. Servers inform the broker about the knowledge they possess by using the advertise performatives. If an intelligent agent needs information from external sources, it sends a request to this broker agent. The broker, which maintains a registry of the agents in the domain, directs the query to the agent that might have enough specialized knowledge to answer the query. This architecture is shown in Figure 3.

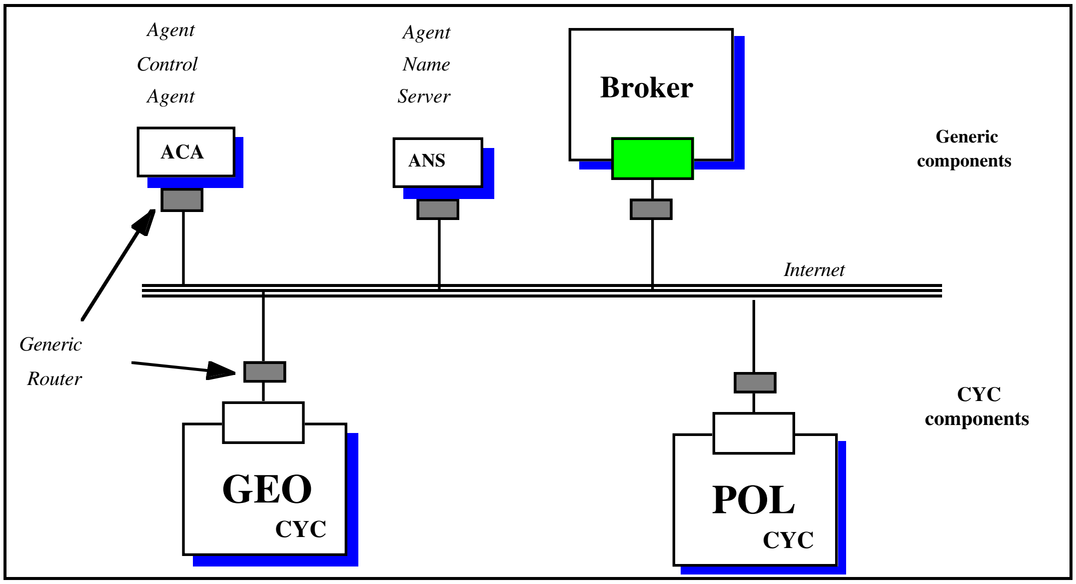

Figure 3. The demonstration architecture.

This intermediate forwarding of request (and later of the reply), called *content-based routing*, has many advantages:

- Individual agents do not have to keep track of what kind of information can be obtained from which agent.
- Information suppliers and consumers can continuously issue and retract advertisements and requests, so information does not tend to become stale.
- Content-based routing helps avoid replication of information, thereby maintaining consistency across the system. In other words, updating information in each agent is unnecessary, because the broker always has access to the most up-to-date information.
- The model is scaleable because the number of agents can increase without significant changes to the agent code.

7. Research Issues

We are currently exploring two main research issues—when should one agent seek help from another and how should two agents interact?

7.1 When should one agent seek help from another?

Because communication costs will dominate computation costs in the distributed environment of CFN, it is important to choose to externally expand a node of the inference tree only when doing so is likely to lead to the desired proof. The approach with which we are currently experimenting is to select a subgoal to be solved externally only after a complete local search; information about the predicates that compose the failed search tree are then used to decide which subgoals should be sent to another agent during a second round of inference. This approach is effectively a form of iterative deepening.

An alternative architecture that is already in place includes knowledge about which agent knows about which domains directly in the Cyc image. Each Cyc agent is given knowledge of what it knows and what other agents know. When a node is selected for expansion, Cyc determines whether it is an expert in that node's predicate. If it is, no external expansion is performed. If it is not, Cyc selects an agent that has the requisite expertise, and determines whether that agent is reachable. In essence we check whether an expert for a literal exists. This approach works well for a small set of tightly-coupled Cyc agents.

Categorizing agents as either ignorant or expert with respect to some body of knowledge is a useful first approximation to characterizing what an agent knows. Another general issue is developing better ways to describe how much an agent knows about a domain. For example, it would be helpful to know the relationship between the knowledge sets of two agents (*e.g.,* that A's knowledge about geography subsumes B's) or the degree of domain knowledge "coverage" that an agent or set of agents offers (*e.g.,* agents A, B and C together have "complete" knowledge about an organization's employees).

Whenever information and knowledge from different sources is to be integrated, inconsistent and contradictory information must be handled. We have not yet addressed these problems (beyond using Cyc's built-in mechanisms for handling contradictions); we currently assume that agents known as experts with respect to a subject contain only valid information and sound reasoning procedures. Dealing with these problems opens up a number of new research issues that will affect when the reasoning help of other agents will be sought, which agents will be selected for interact, and how the information they provide will be interpreted and integrated.

7.2 How should two agents interact?

There are two main problems that fall under the rubric of agent interaction. First, how does an agent decide with which other agents it should communicate? Second, once an appropriate agent has been selected, how should the dialogue proceed?

The solution to the first problem requires the selection of an appropriate agent architecture. We are experimenting with a variety of mediated architectures; Section 6 described our basic approach. The main concern in developing an agent architecture is that it be scaleable.

A critical open issue is how to generate appropriate meta-data descriptions of an agent's knowledge in a way that is useful, automatic, abstractable and easily shared. An agent's meta-data must be useful in helping other agents recognize that it may have information or knowledge that might be relevant. This means that there must be effective procedures for matching a description of one agent's information needs (*e.g.,* a query), with another agent's knowledge and reasoning capabilities. We are currently matching at the level of predicates, which is too fine a granularity for large-scale systems. One promising approach that we are beginning to investigate is the use of information retrieval techniques applied to the symbolic names of knowledge base constants (*e.g.,* names of predicates, functions, relations, attributes, domain individuals and domain values.)[2] It is important to be able to generate these meta-data automatically from the knowledge-base. Indexing or abstracting a

---

[2] These terms might first undergo some kind of syntactic or semantic decoding and might also be augmented by natural language terms associated with their semantics.

knowledge-base by hand or even semi-automatically will certainly not scale. By abstractable, we mean that a meta-data representation should permit representation at different levels of granularity, generality, or precision. Again, this is an important criterion for scalability. Finally, easy of sharing implies that we do not require two agents to be similar or to understand the details of each other's internal representation or reasoning method to use their meta-data.

The solution to the second problem, what kind of dialogs should agents have, involves in large part choosing an appropriate set of conversational primitives, together with conventions for how those primitives will be combined to form valid conversations. Thus far, we have been satisfied with the core primitives provided by KQML (*i.e.,* KQML's performatives); as our architecture grows in complexity we may need to develop new conversational conventions.

More interesting questions remain, however, that involve the nature of the *content* exchanged by agents. The simplest relationship between two agents is a client-server relationship in which one agent sends another a *query* and eventually gets back one or more extensional *answers.* Slightly more complicated is a peer-to-peer relationship in which agents may engage in embedded sub-dialogs. Such conversation is still based on a query and extensional answer model. More flexibility is afforded if agents are allowed to give intensional answers in response to a query. Still more generality comes in allowing agents to reply to a query with a more fully elaborated query which corresponds to a partial proof. Finally, perhaps the most general abstract model of a dialog is one in which the basic exchange involves partially elaborated proof trees—Agent A sends agent B a proof tree, and Agent B extends some portion of the tree and returns it to A.[3]

8. Conclusions

The Cycic Friends Network has been successful in demonstrating that Cyc-based systems can be used in a distributed agent architecture. A set of Cyc-based agents sharing a large common core of knowledge were able to reason together. Each agent contributed some specialized knowledge that the others did not have. The resulting collaboration allowed the agents to solve problems that no subset could. The Cyc agents communicated using the KQML agent communication language with CycL as a content language. This work has led to a variety of interesting questions about how best to coordinate inference in an Internet environment.

9. Acknowledgments

This work has been the result of fruitful collaborations with a number of colleagues with whom we have worked on KQML and other aspects of the Knowledge Sharing Effort. We wish to specifically thank and acknowledge Don McKay, Robin McEntire, Richard Fritzson, Charles Nicholas, R. Scott Cost, Anupama Potluri, Chelliah Thirunavukkarasu and Dionne Warwick.

---

[3] This could be used as a model of "mobile computing," since a partial proof tree is a kind of specification of a non-deterministic declarative program. The proof tree is passed around to various reasoners who have their own local sources of knowledge. This is similar to the manner in which a mobile program moves from platform to platform to execute in environments that contain the data and resources they need.

10. Bibliography

Tim Finin, Yannis Labrou, and James Mayfield, KQML as an agent communication language, invited chapter in Jeff Bradshaw (Ed.), ``Software Agents'', MIT Press, Cambridge, to appear, (1995).

Tim Finin, Anupama Potluri, Chelliah Thirunavukkarasu, Donald McKay and Robin McEntire, *On Agent Domains, Agent Names and Proxy Agents*, Proceedings of the CIKM'95 Workshop on Intelligent Information Agents, Baltimore MD, December 1995.

Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. KQML: an information and knowledge exchange protocol. In International Conference on Building and Sharing of Very Large-Scale Knowledge Bases, December 1993. A version of this paper will appear in Kazuhiro Fuchi and Toshio Yokoi (Ed.), "Knowledge Building and Knowledge Sharing", Ohmsha and IOS Press, 1994. An online copy can be obtained from "http://www.cs.umbc.edu/kqml/papers/kbks.ps".

Tim Finin, Don McKay, Rich Fritzson, and Robin McEntire. The KQML information and knowledge exchange protocol. In Third International Conference on Information and Knowledge Management, November 1994.

Rich Fritzson, Tim Finin, Don McKay and Robin McEntire. KQML—A Language and Protocol for Knowledge and Information Exchange, 13th International Distributed Artificial Intelligence Workshop, July 28-30, 1994. Seattle WA.

Guha, R. V., D. B. Lenat, K. Pittman, D. Pratt, and M. Shepherd. "Cyc: A Midterm Report." Communications of the ACM 33 , no. 8 (August 1990).

Guha, R. V. and D. B. Lenat. "Cyc: A Midterm Report." AI Magazine (Fall 1990).

Yannis Labrou and Tim Finin. A semantics approach for KQML—a general purpose communication language for software agents. In Third International Conference on Information and Knowledge Management, November 1994. Available on-line as http://www.cs.umbc.edu/kqml/papers/kqml-semantics.ps.

Lenat, D. B. and R. V. Guha. Building Large Knowledge Based Systems. Reading, Massachusetts: Addison Wesley, 1990.

Lenat, D. B. and R. V. Guha. "Enabling Agents to Work Together." Communications of the ACM 37, no. 7 (July 1994).

Lenat, D. B. "Steps to Sharing Knowledge." In Toward Very Large Knowledge Bases, edited by N.J.I. Mars. IOS Press, 1995.

Lenat, D. B. "Artificial Intelligence." Scientific American (September 1995).

James Mayfield, Yannis Labrou and Tim Finin. Evaluation of KQML as an Agent Communication Language, the IJCAI-95 Workshop on Agent Theories, Architectures, and Languages.

R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. AI Magazine, 12(3):36-56, Fall 1991.

R. Patil, R. Fikes, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In B. Nebel, C. Rich, and W. Swartout, editors, Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR'92), San Mateo, CA, November 1992. Morgan Kaufmann.