The DARPA Knowledge Sharing Effort: Progress Report

Ramesh S. Patil
USC Info. Sci. Inst.
Marina del Rey, California

Richard E. Fikes
Stanford University
Palo Alto, California

Peter F. Patel-Schneider
AT&T Bell Labs
Murray Hill, New Jersey

Don Mckay
Paramax Systems Corp.
Paoli, Pennsylvania

Tim Finin
Univ. of Maryland
Baltimore, Maryland

Thomas Gruber
Stanford University
Palo Alto, California

Robert Neches
USC Info. Sci. Inst.
Marina del Rey, California

Building knowledge-based systems today usually entails constructing a new knowledge base from scratch. Even if several groups of researchers are working in the same general area, such as medicine or electronic diagnosis, each team must develop its own knowledge base from scratch. The cost of this duplication of effort has been high and will become prohibitive as we build larger and larger systems. Furthermore, lack of methodology for sharing and communicating knowledge poses a significant road-block in developing large multi-center research projects such as DARPA/Rolm Laboratory Planning and Scheduling Initative [21]. To overcome these barrier and advance the state of the art, we must find ways of preserving existing knowledge bases, and sharing, reusing, and building on them.

The Knowledge-Sharing Effort, sponsored by the Defense Advanced Research Projects Agency (DARPA), The Air Force Office of Scientific Research (AFOSR), the Corporation for National Research Initiative (NRI), and the National Science Foundation (NSF), is an initiative to develop the technical infrastructure to support the sharing of knowledge among systems. [27] The goal of this effort is to develop a technology that will enable researchers to develop new systems by selecting components from library of reusable modules and assembling them together. Their effort will be focused on creating specialized knowledge and reasoners specific to the task of their system. Their new system would inter-operate with existing systems, using them to perform some of its reasoning. In this way, declarative knowledge, problem solving techniques and reasoning services could all be shared among systems. The reusable modules in the library them-selves will benefit from refinements that are only possible through extensive use. This would facilitate building larger systems cheaply and reliably. The infrastructure to support such sharing and reuse would lead to greater ubiquity of these systems, potentially transforming the knowledge industry.

The work in the Knowledge-Sharing Effort began with the identification of the impediments to knowledge sharing and corresponding needs for the development of technology to overcome these impediments. Four key areas were identified for the initial effort. They are: (1) mechanisms for translation between knowledge bases represented in different languages; (2) common versions of languages and reasoning modules within families of representational paradigm; (3) protocols for communication between separate knowledge-based modules, as well as between knowledge-based systems and databases; and, (4) libraries of "ontologies," i.e., pre-fabricated foundations for application-specific knowledge bases in a particular topic area.

A detailed discussion of the impediments, and an analysis of the issues that motivated us to focus on these four types, appears in [27]. That article also describes the working groups (comprised of researchers from the DARPA AI community and other volunteers) that were established to address these issues. The next four sections describe the progress made by each of the four working groups in addressing these issues through the development of draft specifications, implementations and experiments.

1 An Interlingua for Knowledge Interchange

For a knowledge-based system to incorporate encoded knowledge from a library or to interchange knowledge with another system, the knowledge must either be represented in the receiving system's representation language or be translatable in some practical way into that language. Since an important means of achieving efficiency in application systems is to use specialized representation languages that directly support the knowledge processing requirements of the application, we cannot expect a standard knowledge representation language to emerge that would be used generally in application systems. Thus, we are confronted with a *heterogeneous language problem*. We may, however, be able to deal with that problem by developing a knowledge interchange language that would be commonly used as an *interlingua* for communicating knowledge between computer programs.

Given such an interlingua, a sending system could translate knowledge from its application-specific representation into the interlingua for communication purposes and a receiving system could translate knowledge from the interlingua into its application-specific representation before use. In addition, the interlingua could be the language in which libraries would provide reusable knowledge bases. An interlingua eases the translation problem in that without an interlingua one must write N pairs of translators in order to communicate knowledge to and from N other languages. With an interlingua, one need only write one pair of translators into and out of the interlingua.

1.1 KIF – a Knowledge Interchange Format

The Interlingua Working Group, chaired by Richard Fikes and Michael Genesereth, is attacking the heterogeneous language problem by developing and testing a language for use as an interlingua called the Knowledge Interchange Format (KIF)[16]. The group began its work by observing that an interlingua needs to be a language with the following general properties:

- A formally defined declarative semantics;
- Sufficient expressive power to represent the declarative knowledge contained in typical application system knowledge bases; and
- A structure that enables semi-automatic translation into and out of typical representation languages.

The working group then merged ongoing language design efforts to produce a preliminary version of the KIF language which could be used as a straw man interlingua in knowledge interchange experiments and design discussions. Since then, the language has been continually evolved and expanded based on feedback from ongoing e-mail discussions, formal design reviews, translation of example knowledge bases, and interoperation experiments.

KIF is an extended version of first order predicate logic. The current 3.0 version of KIF has the following features:

- Simple list-based linear ASCII syntax suitable for transmission on serial media. For example, the following is a KIF sentence:

```
(forall ?x (=> (P ?x) (Q ?x)))
```

- Model-theoretic semantics with axiomatic characterization of a large vocabulary of object, function, and relation constants.
- Function and relation vocabulary for numbers, sets, and lists.
- Support for expression of knowledge about the properties of functions and relations. Functions and relations are included in the universe of discourse as sets of lists so that they can be arguments to relations (e.g, `transitive` and `one-one`) and functions (e.g., `inverse` and `range`). In addition, a `holds` relation is included that is true when its first argument denotes a relation that has as a member the list consisting of the items denoted by the remaining arguments. So, for example, one could define transitivity as follows:

```
(<=> (transitive ?r)
     (=> (holds ?r ?x ?y)
         (holds ?r ?y ?z)
         (holds ?r ?x ?z)))
```

- A sublanguage for defining objects, n-ary relations, and n-ary functions that enables augmentation of the representational vocabulary and specification of domain ontologies. Definitions can be *complete* in that they specify an equivalent expression or *partial* in that they specify an axiom that restricts the possible denotations of the constant being defined. For example, the following is a complete definition of the unary relation `bachelor`:

```
(defrelation bachelor (?x) :=
   (and (man ?x) (not (married ?x))))
```

and the following is a partial definition of a binary relation `above` which specifies that `above` is transitive and holds only for "located objects":

```
(defrelation above (?b1 ?b2)
   :=> (and (located-object ?b1)
            (located-object ?b2))
   :axiom (transitive above))
```

- Support for expression of knowledge about knowledge. KIF expressions are included as objects (i.e., lists) in the universe of discourse, and functions are available for changing level of denotation. For example, the following sentence says that Lisa has the same belief as John about the material of which things are made:

```
(=> (believes john '(material ,?x ,?y))
    (believes lisa '(material ,?x ,?y))
```

and the following sentence says that every sentence of the form (=> ϕ ϕ) is true:

```
(=> (sentence ?p) (true '(=> ,?p ,?p)))
```

- A sublanguage for stating both monotonic and nonmonotonic inference rules. For example:

```
(<<= (flies ?x)
     (bird ?x) (consis (flies ?x)))
```

A KIF reference manual describing the entire language in detail is available through anonymous FTP from `hudson.stanford.edu`[17]. The working group expects the current language design to remain relatively stable and for future versions to be essentially extensions to the existing language. Extensions under active consideration include support for uncertain knowledge and contexts, and additional support for default knowledge.

KIF is intended to be a core language which is expandable by defining additional representational primitives. For example, one can define a frame language vocabulary of classes, slots, number restrictions, value restrictions, etc. (as Gruber has done in [19]) so that knowledge can be expressed in a form directly analogous to a frame language. Thus, given suitable definitions, one could define a "guest meal" as being a meal in which there is at least one guest and the food is gourmet as follows:

```
(defrelation guest-meal (?m)
   :=> (and (meal ?m)
            (at-least-fillers ?m guest 1)
            (all-fillers ?m food
                            gourmet-food)))
```

1.2 Knowledge Interchange Experiments using KIF

The problems involved in interchanging knowledge bases are not yet well understood, and there is open debate as to whether a generally useful interlingua can be specified. The Interlingua Working Group is attempting to inform that debate by developing KIF as a candidate interlingua and by promoting knowledge interchange experiments designed to substantially test the viability and adequacy of KIF as an interlingua. Several small scale experiments have been conducted thus far and multiple projects are underway to build and test KIF translators. These activities, though still in preliminary stages, have already been very productive in identifying issues that need to be resolved and technology that needs to be developed in order for knowledge interchange to be a practical reality. We describe three examples of such activities below.

Ramesh Patil built translators to an early version of KIF from CLASSIC [4] and from LOOM [22]. He then used those translators to produce KIF versions of simple CLASSIC and LOOM knowledge bases. As expected, such translation experiments highlighted weaknesses in KIF and motivated evolution of the language. In general, producing KIF translations of a wide range of sample knowledge bases is an effective means of evaluating the expressive adequacy of KIF and focusing its continuing development. Building the translators themselves does not appear to be problematical. The primary issue is whether KIF has sufficient expressive power to represent the declarative knowledge expressible in the source language.

Translating knowledge *out of* KIF is in general an intractable problem because any given proposition can be expressed in KIF in many equivalent but syntactically different forms and the recognition grammar for a target language will only be able to recognize some subset of those forms. The translation task, therefore, involves applying equivalence preserving rewrite rules to transform unrecognizable sentences into recognizable forms. Despite the worst-case complexity of logically complete translation, effective translation may be achievable in most situations by logically incomplete techniques combined with interactive direction from the user. To explore that hypothesis, Fikes and Van Baalen are building a translator development "shell" which will contain a grammar-based recognizer, a goal-directed rewrite rule interpreter, a library of general-purpose rewrite rules, facilities for hand translation of problematic sentences, etc. [12]. Initial versions of the basic components of that shell have been implemented and have been used to successfully translate simple KIF knowledge bases into CLASSIC.

A knowledge interchange capability is important both to enable *incorporation* of knowledge into a knowledge-based system (e.g., during system development) and to enable *interoperation* of knowledge-based systems so that they can cooperatively perform tasks and solve problems. KIF is being used as the knowledge level inter-agent communication language in multiple interoperation experiments, including those conducted by Mike Genesereth using the Designworld system [15] and those being conducted by participants in the Palo Alto Collaborative Testbed (PACT).

Designworld is an automated prototyping system for small scale electronic circuits built from standard parts (TTL chips and connectors on prototyping boards). The design for a product is entered into the system via a multi-media design workstation; the product is built by a dedicated robotic cell; and, if necessary, the product, once built, can be returned to the system for diagnosis and repair. The system consists of eighteen processes on six different machines. Each of the eighteen programs is implemented as a distinct agent that communicates with its peers via messages in a KQML-like Agent Communication Language (ACL) that uses KIF as the "content" language.

PACT is a laboratory for exploring the use of knowledge sharing technology and agent-based system integration architectures to support concurrent engineering. Participants include research groups at Stanford University, Lockheed AI Laboratory, Hewlett-Packard Laboratories, and Enterprise Integration Technologies. The initial experiments integrated four preexisting concurrent engineering systems into a common computational framework and explored engineering knowledge exchange in the context of a distributed simulation and simple incremental redesign scenario [9]. In those experiments, each of the individual systems was used to model one or more components of an example programmable electro-mechanical device, a small robotic manipulator. The systems interact via software agents which use KQML as the "performative" language and KIF as the "content" language during knowledge interchange.

Although these experiments have not yet placed severe demands on KIF as an interlingua, KIF successfully provided what was needed, namely a clearly specified logical sentence language for interchange of assertions, queries, and simulation inputs and outputs.

2 The Knowledge Representation System Specification

Even within a single family of knowledge representation systems (e.g. KL-ONE) minor differences in syntax and semantics between systems pose significant barriers to knowledge sharing. The goal of the Knowledge Representation System Specification (KRSS) group is to develop common specifications for the representational component of families of knowledge representation systems. These specifications will help facilitate the transfer of collections of knowledge between knowledge representation systems in the same family, by reducing the representational differences among systems in the family. The intent of the group is to produce, by-and-large, descriptive specifications, although reconciliation of some syntactic differences will almost certainly be required.

Specifications produced by the group will concentrate on the representational components of the family of knowledge representation systems. Thus, they will provide a complete definition of the representation language underlying the family, but will not include a complete definition of the interface functions that are required in a useful knowledge representation system. Instead the specifications will only define a minimal interface, one that is sufficient to create knowledge bases and query them in limited ways. Also, specifications will completely ignore user-interface issues.

These specifications will definitely not be interlinguas. The representation formalism in the specifications will be specific to the family of representation systems under consideration, and will not be general-purpose representation logics. The specifications also have to be concerned with the computational properties of the formalism they define (i.e., how hard inference in the formalism is), as the aim of the group is to specify knowledge representation *systems*, and not just abstract formalisms.

The initial effort of the KRSS group is the development of a specification for knowledge representation systems based on what are now called description logics (also known as frame-based description languages, terminological logics, etc.). These systems include BACK [31], CLASSIC [6], KRIS [2], and LOOM [22]. This group of systems was chosen partly because there is a large number of systems that are based on description logics (see above), partly because there was already some interest in the community of developers of such systems for a common specification [1], partly because many of the people in the initial group gathered together at the start of the DARPA Knowledge Sharing Initiative were working with such systems, and partly because such systems have a formal basis that is readily amenable to a well-defined specification. There has also been considerable study of the formal properties of reasoning in systems based on description logics. This includes studies of how reasoning should proceed in such systems [26] and the computational complexity and decidability of reasoning in description logics [5, 25, 10, 30]. The presence of such a large body of formal work makes the specification process much easier.

Although there is a common background for all knowledge representation systems based on description logics, there is surprising variance in several dimensions in the systems. First, different systems have different input syntaxes. One goal of the initial KRSS effort is to minimize differences in this dimension. Second, different systems have different interfaces, both functional and user interfaces. Another goal of the initial KRSS effort is to minimize differences in the portion of the functional interface used to construct and directly query knowledge bases. However, the rest of the interfaces of the various systems will not be incorporated into the specification, as it is outside the goals of the KRSS group.

The main difference between the various systems is that they take different positions in the trade-offs among expressive power, completeness of inference, and resource consumption. Some systems try to be as complete as possible in a less-expressive description logic while consuming as few resources as possible, trading off expressive power for computational benefits. Some systems implement complete inference in a moderately-expressive but decidable description logic, trading off possible resource consumption for better expressive power. Some systems implement only partial inference in an expressively-powerful description logic, trading off completeness for expressive power.

Many points in this set of trade-offs are reasonable, so a specification has to allow for both the current set of trade-offs, and also for possible future trade-offs. This means that the specification will not be a complete specification nor even a nearly complete specification.

The approach that has been taken in the specification is to define an expressively powerful description logic, including both a syntax and a semantics, incorporating those constructs whose meaning has been generally agreed upon by the community. Along with the description logic is a set of interface functions that allow for the construction, manipulation, and querying of description-logic knowledge bases. These functions allow

- the formation of descriptions and sentences;
- the definition of concepts and roles from descriptions;
- the assertion of sentences, including ground facts about individuals and simple rules about concepts;
- the creation of individuals and reasoning about their identity;
- the making of local closed-world statements;
- the making of default statements about instances of concepts;
- the retracting of previously-told assertions; and
- the querying of knowledge bases.

The non-query functions are defined by their effect on an abstract knowledge base, which is a collection of statements in the description logic. The results of the query functions are (mostly) defined by semantic relationships between the knowledge base and the query.

Because it is impossible to efficiently perform inference in the full description logic, conforming systems are not required to completely implement it. Conforming systems are free to recognize only a subset of the syntax of the logic, and need not even perform complete reasoning in the subset that they do recognize. However, such systems must use this logic as the ideal meaning of their knowledge bases, and must perform "sound" reasoning with respect to the logic.

Conforming systems are not completely free in what portion of the logic they choose to address. There is a core portion of the logic that all conforming systems are required to implement; in this way a minimal competence is required for all conforming systems. The core is not just a syntactic subset of the full logic—complete inference on even very minimal subsets of the logic is very difficult—but is instead a set of constructs that must be recognized, along with a set of inferences that must be performed on these constructs.

Most of the debate on the specification has involved the details of this core. The constructs to include in the core, the inferences to perform on them, and how to specify these inferences have all been subjects of debate. This was to be expected, as the specification of the core is where the specification is making decisions on matters that have been decided in different ways by different systems. Devising a core that is both reasonable and non-trivial is an interesting exercise in how to balance various representation and implementation concerns.

There is now (July 1992) a second draft of the complete specification that has been distributed to interested parties. Some changes still need to be made to this draft. First, formal work in description logics has advanced, and should be incorporated into the specification. Second, there are portions of the draft, particularly in the inferences required in the core, that are objectionable to some parties. By September 1992, there should be a third draft prepared and discussed, and by the end of October 1992 a final version of the specification should be available. Also, a method for demonstrating compliance with the specification will be developed.

Future work in the KRSS group effort on description-logic based systems will then consist of augmenting the specification as new formal work on description logic produces relevant results and as new implementation techniques make it possible to extend the core. Also, other families of knowledge representation systems may be given the same treatment, provided that developers are interested.

3 Knowledge Query And Manipulation Language (KQML)

The *External Interfaces* working group was originally charged with addressing the general problem of defining standard high-level interfaces for knowledge representation systems. This was seen as including such diverse interfaces as those to other KR systems, DBMSs, active sensors, and human users. Over the past two years, this working group has focused on and experimented with a somewhat narrowed and more focused problem – designing a common high-level language (KQML) and associated protocol which can be used by software systems for the run-time sharing of information and knowledge. This section briefly describes the current status of the effort to specify KQML and experiment with its use in several testbeds.

3.1 Overview

The Knowledge Query and Manipulation Language (KQML) is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. KQML can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving. KQML focuses on an extensible set of *performatives*, which defines the permissible operations that agents may attempt on each other's knowledge and goal stores. The performatives comprise a substrate on which to develop higher-level models of interagent interaction such as contract nets and negotiation [8, 33].

In addition, KQML provides a basic architecture for knowledge sharing through a special class of agent

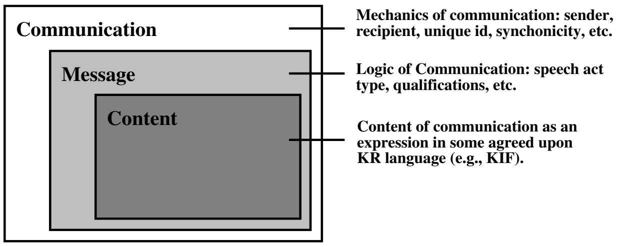

Figure 1: KQML expressions can be thought of as consisting of a content expression encapsulated in a message wrapper which is in turn encapsulated in a communication wrapper.

called *communication facilitators*. These agents coordinate the interactions of other agents by providing such functions as:

- identification of other agents with which to communicate both explicitly via "names" or "addresses" or implicitly via declared topics of interest or capabilities,
- maintaining registration databases of knowledge services offered and sought by agents,
- communication services (e.g., forwarding information from one agent to other interested agents), and
- content translation to bridge semantic and ontologic differences between end agents.

These functions are embodied in special performatives (which take messages as arguments), and in the way that facilitators treat messages received from application agents.

The ideas which underly the evolving design of KQML are currently being explored through experimental prototype systems which are being used to support two testbeds: the Palo Alto Collaborative Testbed (PACT) [9] which is focused in the concurrent engineering domain, and the DARPA/Rome Planning Initiative (DRPI) which deals with military transportation planning [13].

3.2 KQML Expressions are Layered

KQML expressions consist of a content expression encapsulated in a message wrapper which is in turn encapsulated in a communication wrapper, as shown in Figure 1. Thus the language is thought of as being divided into three layers: content, message and communication. The content layer contains an expression in some language which encodes the knowledge to be conveyed. The format of this expression is unimportant to KQML; it can carry any type of content expressed in any representation language which follows some general syntactic constraints (currently, the content expression must be an s-expression). However, there are emerging conventions for knowledge representation (e.g., Interlingua, KIF [17], etc) and standards for *persistent objects* (e.g., the OMG Object Request Broker) which may prove to be very valuable in the near future.

The primary purpose of the message layer is to identify the speech act or performative that the sender attaches to the content, such as an assertion, a query or a command, and any of a small set of qualifiers that may be appropriate to that performative. In addition, since the the content is opaque to KQML, this layer also includes optional features describing the content's language, the ontology it assumes and a descriptor naming a topic within the ontology. These features make it possible for the protocol implementation to analyze, route and properly deliver messages even though their content may be inaccessible.

The final communication level adds a second layer of features to the message which describe the lower level communication parameters, such as the identity of the sender and recipient, a unique identifier associated with the communication and whether the communication is meant to be synchronous or asynchronous. These are used by the network layer which provides reliable transfer of bytes between processes on a network.

3.3 KQML Performatives

The message layer is used to encode a message that one application would like to have transmitted to another application and forms the core of the language, determining the kinds of interactions one can have with a KQML-speaking agent. It can be thought of as a "speech act layer", since an important attributes to specify about the content is what kind of "speech act" it represents – an assertion, a query, a response, an error message, etc.

Structure. Conceptually, a KQML message consists of an operator or *performative*, its associated arguments which constitute the real *content* of the message and a set of optional arguments which describe the content in a standard, language-independent manner. For example, a message representing a query about the location of an particular airport might be encoded as:

```
(ask (geoloc lax (?long ?lat))
     :number_answers 1
     :ontology drpi_geo)
```

In this message, the KQML performative is *ask*, the content (i.e., knowledge being sought) is $(geoloclax(?long?lat))$, the number of answers requested is 1, the language in which the content is expressed is (by default) *kif* and the ontology to be assumed is that named by the token $drpi_geo$. The same general query could be conveyed in using standard Prolog as the content language in a form that requests the set of all answers as:

```
(ask "geoloc(lax, (Long,Lat))"
     :language standard_Prolog
     :number_answers all
     :ontology drpi_geo)
```

Semantics. It is our intention to allow the set of KQML performatives to be extensible. We will identify a core set of performatives that will have a well defined meaning. An KQML-speaking agent need not implement or handle all of the performatives in this core, but for those it does, it must adhere to the standard semantics. Moreover, it is our goal to provide a standard mechanism by which one can define the semantics of new performatives, allowing the set to be extended. The semantics of the core performatives will be defined in terms of a smaller set of *primitive performatives*. The semantics of these primitive performatives are defined with respect to a simple and general model of agents in which each agent as a store of information structures (i.e., "belief" like items) and a store of goals structures (i.e., items which may effect the agent's future behavior).

Primitive Performatives. We are currently working with a set of four primitive performatives from which we believe the core and various interesting extensions can be defined. These four primitives provide operators to present an agent with items to add ($ADVISE$) and remove ($UNADVISE$) from its information store and to add ($ACHIEVE$) and remove from ($FORGET$) its goal store. These four performatives are primarily used as a means to specify the semantics of the larger core performatives.

Core Performatives. The core set of performatives is expected to include several dozen operators which most KQML-speaking agents will support. If an agent accepts a message with a core performative, it must adhere to its agreed upon semantics. Some of these performatives will accept optional arguments which serve as qualifier. Figure 2 shows some examples of performatives that are in the current specification.

Messaging via Facilitators. Any substantial collection of interacting agents will require some structure on information flow [20, 28, 32]. For this reason, KQML introduces a class of communication facilitator agents that help manage the message traffic among application agents. Facilitator agents can route performatives to appropriate agents (MONITOR performatives in particular), record the performative-processing abilities of new agents, and bridge the capabilities of superficially incompatible agents (through buffering, translation, and problem decomposition). These facilitation functions will be reflected in new core performatives, e.g., **(FORWARD** *agent-name message*) and **(DISTRIBUTE** *message*).

- **(ASSERT P)** - Add P to the agent's information store, performing whatever reasoning the agent can perform.
- **(RETRACT P)** - Remove P from the agent's information store if present, signalling an error if not present and performing whatever reasoning the agent can perform.
- **(ASK P)** - Query the agent's information store to find answers matching query P. The number of answers returned is governed by an optional argument.
- **(GENERATOR P)** - Reply with a *generator* that the recipient can use to elicit a stream of answers to the query P.
- **(MONITOR P)** - Modify the agent's goal store to cause it to inform the sender whenever a sentence matching P becomes true.

Figure 2: These are a few of the core KQML performatives.

Software Architecture. As Figure 3 shows, a typical KQML-speaking agent will be built using two reusable pieces – an interface between the agent's system language (e.g., LOOM or Prolog) which ties communication actions to system actions, and a router which handles the low-level communication chores necessary to talk to other agents. These might all be done within a single process (e.g., in Lisp) or might include several processes (e.g., the router might be done in C or Perl).

3.4 Status and Open Issues

The ideas which underly the evolving design of KQML are currently being explored through experimental prototype systems which are being used to support two testbeds: the Palo Alto Collaborative Testbed (PACT) [9] which is focused in the concurrent engineering domain, and the DARPA/Rome Planning Ini-

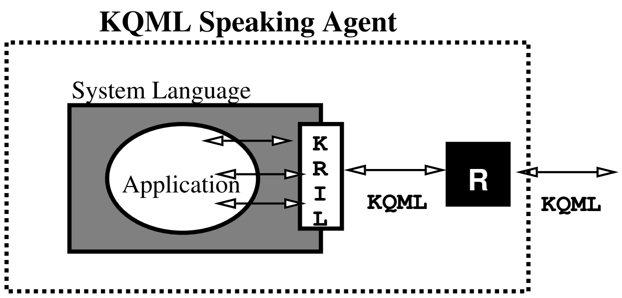

Figure 3: A typical KQML-speaking agent will be built using two reusable pieces – an interface between the agent's system language (e.g., LOOM or Prolog) which ties communication actions to system actions, and a router which handles the low-level communication chores necessary to talk to other agents.

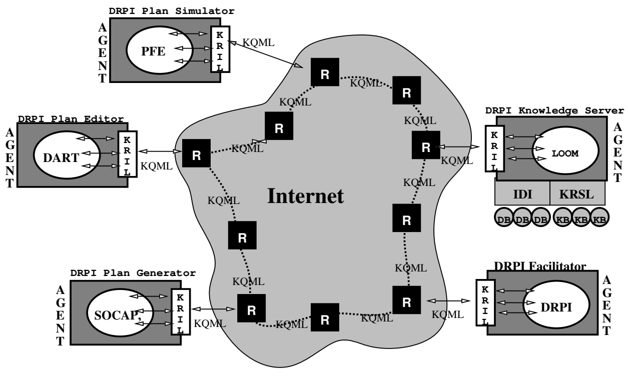

Figure 4: KQML will be used as communication language among the various agents which make up the DRPI testbed. It will be used, for example, to support the interchange of knowledge among the planner, the plan simulator, the plan editor and the DRPI knowledge server which is the repository for the shared ontology and access point for common databases.

tiative (DRPI) which deals with military transportation planning.

KQML use in PACT. The Palo Alto Collaborative Testbed (PACT) uses KQML as its medium for agent interaction in support of concurrent engineering. PACT participants modified several existing knowledge-based engineering systems to speak KQML and thereby exchange design and manufacturing knowledge of mutual interest. (For example, the mechanical modeler sends the controls modeler knowledge regarding the dynamics of the design; the power modeler sends the manufacturing process planner knowledge regarding a motor replacement.) These agents find each other in part through facilitators, which handle message forwarding, content-based routing, and simple format translations.

KQML use in DRPI. The DARPA/Rome Planning Initiative is using KQML as the communication language among the various agents that make up the testbed and feasibility demonstrations. Figure 4 shows KQML being used, for example, to support the interchange of knowledge among the planner, the plan simulator, the plan editor and the DRPI knowledge server, which is the repository for the shared ontology [21] and access point to common databases through the *Intelligent Database Interface* [23, 29]

Open Issues. The design of KQML has continued to evolve as the ideas are explored and feedback is received from the prototypes and the attempts to use them in real testbed situations. We mention here a few of the important issues that we expect to be addressed in the coming year.

The core set of performatives is still undergoing revision as we experiment with its use. This set needs to be stabilized and well specified. In particular, we need to refine the model of what a communication facilitator is and what services it might offer so that we develop a good set of performatives to support their effective use.

A method for defining new extensions to the core set needs to be worked out. This includes a method for defining them for humans as well as a method to allow one agent to define a new performative to another.

The basic model of a knowledge representation agent that we have been working with is quite simple. One of several extensions that may be needed, for example, is a mechanism to define contexts within an agents information and goal stores.

An important part of KQML will be the protocols associated with the different performatives. There are some general issues which go beyond defining the semantics of particular performatives that must be addressed. These general protocols include such things as refusing to accept a message, error reporting, security, and transaction oriented processing.

4 Shared, Reusable Knowledge Bases

The SRKB Working Group (Shared, Reusable Knowledge Bases) of the DARPA Knowledge effort is working on the problem of sharing the *content* of formally represented knowledge. Sharing content requires more than a formalism (KIF) and communication protocol (KQML). Of course, understanding the nature of what needs to be held in common between communicating agents, or between the author of a book and its reader, is a fundamental question for philosophy and science. The SRKB group is focusing on the practical problem of building knowledge-based software that can be shared and reused as off-the-shelf technology. The charter of the group is to identify the technical barriers to the sharing and reuse of formally represented knowledge by AI programs, and to provide a forum for experimentation with possible approaches.

4.1 Strategy: Common Ontologies as a Sharing Mechanism

The strategy is to focus on common ontology as the sharing mechanism [27, 18]. What is a common ontology? Every knowledge-based system is based on some conceptualization of the world: those objects, processes, qualities, distinctions, and relationships that matter for performing some task. A program (or its programmer) makes ontological commitments to a conceptualization by embodying these concepts, distinctions, etc. in a formal representation and using knowledge formulated in that representation during problem solving. By **common ontology** we mean an explicit specification of a the ontological commitments of a set of programs. Such a specification is an objective description—interpretable outside of the programs—of the concepts and relationships that the programs assume and use when interacting with other programs, knowledge bases, and human users.

Operationally, a common ontology can be specified as a set of definitions of representational terms used to construct expressions in a knowledge base, such as classes, relations, slots, and object constants. To make a common ontology shareable, the definitions should consist of human-readable text and machine-enforceable, declarative constraints (i.e., axioms) on the well-formed use of the terminology. The set of terms in a common ontology need not include all the terms used internally in participating programs. Rather, the shared vocabulary defined in a ontology is used for specifying the coupling between programs and knowledge bases (at design time) and for knowledge-level communication among agents (at run time). We hope to enable large-scale sharing and reuse of knowledge bases and knowledge based systems by making common ontologies available as open specifications, much like interchange formats and communication protocols.

The initial activities of the working group have been to explore the research issues in knowledge sharing, and to identify areas where it might be practical and useful to specify common ontologies. The Summer Ontology Project, held at Stanford in 1990, studied the collaborative, multi-disciplinary development of reusable ontologies for describing electromechanical devices and their designs. One outcome was the observation that several approaches to device modeling, from digital circuit modeling to rigid body dynamics, seemed to make commitments to lumped-element models of physical devices. In a lumped-element model, the behavior of a device is described in terms of values of functions (state variables) that map a single independent variable (e.g., time, but not space) to physical quantities (position, force, etc.). A preliminary ontology was proposed to formalize these concepts.

In March of 1991, the SRKB group met at Pajaro Dunes to characterize some of the research issues. There was some controversy about whether it is premature to "standardize" ontologies of any sort, especially those designed to be comprehensive over tasks and domains. Instead, a series of collaborative, grassroots experiments were proposed, in which two or more research groups identify potential candidates for knowledge sharing.

In the past year, several collaborations have begun, and a set of ad hoc subgroups have been formed to study these ontological niches. Each subgroup is tasked with identifying, collecting, making available, and analyzing ontologies for knowledge sharing. We will describe the efforts of these groups within a framework of models of sharing and reuse.

4.2 Models of Knowledge Sharing and Reuse

Three models of sharing and reuse are being explored, and in each, common ontologies play an enabling role.

First is the **library model**, in which bodies of formally represented knowledge are available as off-the-shelf products, like books in a library. In this model, knowledge bases are designed artifacts, and the role of SRKB to help make them available and reusable.

Two ad hoc subgroups are currently active within the library model of sharing. One is an effort by representatives of projects in qualitative physics to specify a common language for model fragments. Model fragments are conceptual building blocks for programs that formulate and assemble engineering models of device behavior, using techniques such as compositional modeling [11]. For example, idealized components such as resistors and physical processes such as liquid flow are represented by model fragments, which are composed to produce simulation models of complete systems. The language under development is a unification of model formulation and simulation systems such as QPE, DME, and QPC, and should enable a community library of model fragments that can be directly executed by these systems. The axiomatic semantics of the language will be expressed in KIF, and the ontological commitments of these programs will be specified as an ontology.

A second subgroup, following up on the Summer Ontology Project, is developing a family of ontologies for specifying various styles of engineering modeling. It is formalizing the classes of algebras used in constraints (e.g., with or without differential equations; qualitative operators), the assumptions underlying component/connection topologies, and the various styles of dynamics analysis (e.g., Newtonian, LaGrangian, Kane's method). This work is complementary to the composition modeling effort; any of these styles of modeling can be formulated using the model fragment language.

A preliminary finding is that the ontological commitments of a given approach to modeling may be factored into separate ontologies. These ontologies form an inclusion hierarchy, where each ontology can inherit (by set inclusion) the definitions of included ontologies. For example, the original proposal for a lumped-element ontology has since been divided into several ontologies, including *continuous-state-space* (commits to describing behavior using state variables) and *hierarchical-component-assembly* (objects are structured into components related by connections and part-of relations). To specify how state variables are associated with components, one writes a third ontology that includes the other two, adding a few additional constraints. To support this sort of modular partitioning of ontologies, the interlingua committee is considering context mechanisms such as Cyc's microtheories.

A second mode of sharing and reuse under investigation is the **software engineering model**. A standard approach to making software reusable is to decompose complex programs into modular pieces, and to provide a formal specification of the inputs, outputs, and function computed by each piece. Knowledge-based systems are like other software in this respect, except that they operate on a special input called the "background knowledge base" or "domain theory." Reusable modules are designed so that the same code can be used on several knowledge bases. However, to write these knowledge bases the developer needs to understand the ontological assumptions and commitments made in the code. An ontology that defines the vocabulary with which to write the knowledge bases can help determine which software module to use on a given problem, how to provide it the necessary domain knowledge, and whether the knowledge base meets the input requirements of the software.

A significant effort is under way in the knowledge acquisition community to formally characterize the tasks being performed by knowledge based systems, and to design modular problem-solving methods that can be combined to address these tasks [24]. For example, complex, amorphous tasks such as diagnosis and planning have been decomposed into more generic subtasks that can be solved with reusable methods such as simple classification, abductive assembly, and varieties of constraint satisfaction. An subgroup led by Mark Musen is studying ways to describe these tasks and methods, and has begun to define ontologies that specify the input and output assumptions of reusable methods.

A second subgroup, headed by Ed Hovy and Doug Skuce, is identifying and analyzing the comprehensive, *top-level* ontologies that are intended to be general across domains and tasks. A motivating application for such ontologies is natural language processing. NLP techniques needs a way to couple to domain knowledge bases (for something to talk about) without committing the programs to particular subject matter areas. For example, the Penman language generation system's "Upper Structure" ontology [3] divides the world up according to the major type distinctions made in English and German (Objects of various types, Processes and Relations of various types, Qualities, etc.). A developer customizes Penman to a particular application domain by defining the domain's concepts as specializations of the appropriate Upper Structure concepts. As a result, the domain concepts inherit the necessary linguistic annotations from their Upper Structure ancestors. In general, such top-level ontologies can be viewed as a software reuse mechanism for programs parameterized by large knowledge bases.

Another subgroup is looking at ontologies that specify semiformal representations of decision making and design rationale (Jeff Bradshaw, Jin Tae Lee, and Charles Petrie). In semiformal rationale support systems, users organize text describing design decisions in to a hypertext document that supports a fixed vocabulary of node types (classes) and link types (relations). For example, in the gIBIS ontology [7], decisions are described in terms of *issues*, *arguments*, and *positions*, and these node types are linked by relations such as *supports* and *objects-to*. The documents structured by these terms are called semiformal or semistructured, since only the node and link types are machine interpretable and the contents of the nodes are not formalized. Several methodologies for developing semiformal documents, and tools to support them, are based on these ontologies of node and link types.

A third kind of sharing and reuse is the **reference model**, typically used to define an integration framework for a family of application programs. A reference model defines the concepts in a domain and/or problem area that are common to the set of application tasks. For example, a reference model for digital circuits includes a formalism for describing the netlist, which is a representation of circuit topology. The reference model ontology commits the participating tools to the existence of shared objects such as components connected by ports in a netlist; this is necessary to enable tools to exchange data.

An international standards effort called PDES/STEP is working on a family of reference-model ontologies for product data, starting by defining primitives for geometry and working toward high level descriptions of behavior and functionality. The DARPA knowledge sharing effort is exploring avenues for collaboration with the PDES organization.

Within the SRKB working group, ad hoc subgroups are studying reference-model ontologies for user interface toolkits (Jim Foley and Bob Neches), manufacturing enterprise models (Mark Fox), and planning/scheduling (Don McKay, Masahiro Hori).

4.3 Technical Support for Ontologies – Ontolingua

Each of the subgroups of the SRKB are charged with identifying and collecting ontologies, and making them available in a form amenable to analysis and possible reuse. However, existing ontologies are either incompletely formalized or written in a specific knowledge representation tool. To address this problem, a system called Ontolingua has been developed [19]. Ontolingua is a mechanism for defining ontologies *portably*, that is, independent of specific representation systems. It allows the definition of classes, relations, and distinguished objects using KIF sentences, and translates these definitions into several implemented representation systems.

Ontolingua's design demonstrates the use of a common ontology to facilitate sharing and reuse (in this case, of ontologies). Translation from a very expressive language (KIF) into restricted languages is inherently incomplete. Therefore, Ontolingua supports a subset of legal sentences that can be translated into a class of commonly-used representation systems: the object-centered or frame-based systems. These implemented systems commit to particular ways of organizing and specifying knowledge about objects, such as inheritance hierarchies and slot descriptions. These ontological commitments are captured in the Frame Ontology, which defines a vocabulary for describing classes, binary relations, and second-order relationships among them (e.g., subclass, instance, class partitions, slot-value restrictions). Ontolingua recognizes the use of Frame Ontology concepts in KIF sentences, and translates them into the special syntax of each target representation system. The Frame Ontology, on top of a syntactically restricted KIF, defines a language for portable ontologies. The Ontolingua software operationalizes the language by providing automatic translation into implemented representation systems.

5 Summary

Moving beyond the capabilities of current knowledge-based systems will require development of knowledge bases that are substantially larger than those we have today. It will require knowledge-based systems to communicate with other knowledge-based systems and conventional software systems in carrying out their functions. Meeting these challenges on a broad scale will require development new knowledge-sharing technology and shared conventions. The on-going efforts in the Knowledge Sharing Effort represent steps in this directions. The efforts underway are neither complete nor comprehensive – they represent an initial first steps that will result in valuable experience and understanding, will identify shortcomings in current methods and point to new research directions, will encourage others to focus on solving problems encountered in knowledge sharing, to explore alternatives and to enhance the state of the art.

Acknowledgments

This effort is supported by NSF grant IRI-9006923, DARPA/NASA-Ames contract NCC 2-719, and a cooperative agreement between USC/ISI and the Corporation for National Research Initiative. We would also like to acknowledge members of the Knowledge Sharing Effort, too numerous to name individually.

References

[1] Franz Baader, Hans-Jürgen Bürckert, Jochen Heinsohn, Bernhard Hollunder, Jürgen Müller, Bernhard Nebel, Werner Nutt, and Hans-Jürgen Profitlich. Terminological knowledge representation: A proposal for a terminological logic. A DFKI note., June 1991.

[2] Franz Baader and Bernhard Hollunder. KRIS: Knowledge Representation and Inference System—system description. Technical Memo TM-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz, November 1990.

[3] John A. Bateman. Upper modeling: A general organization of knowledge for natural language processing. Penman development note, USC/Information Sciences Institute, 1989.

[4] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, 1989.

[5] Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 34–37, Austin, Texas, August 1984. American Association for Artificial Intelligence.

[6] Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alex Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In Sowa [34], pages 401–456.

[7] Jeff Conklin and M. L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. In *Proceedings of the 1988 Conference on Computer Supported Cooperative Work (CSCW-88)*, pages 140–152, Portland, Oregon, 1988. ACM.

[8] Susan E. Conry, Robert A. Meyer, and Victor R. Lesser. Multistage negotiation in distributed planning. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 367–384. Morgan Kaufman, 1988.

[9] Mark Cutkosky, Robert Engelmore, Richard Fikes, Thomas Gruber, Micheal Genesereth, William Mark, Jay Tenenbaum, and Jay Weber. Pact: An experiment in integrating concurrent engineering systems. *IEEE Computer*, 1992. To appear in a special issue on computer-supported concurrent engineering.

[10] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 151–162. Morgan Kaufmann, May 1991.

[11] Brian Falkenhainer and Ken Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

[12] Richard Fikes, Mark Cutkosky, Tom Gruber, and Jeffrey Van Baalen. Knowledge sharing technology project overview. Technical Report KSL 91-71, Stanford University, Knowledge Systems Laboratory, 1991.

[13] T. Finin, R. Fritzson, and D. McKay et. al. A language and protocol to support intelligent agent interoperability. In *Proceedings of the CE & CALS Washington '92 Conference*, June 1992.

[14] T. Finin, R. Fritzson, and D. McKay et. al. An overview of KQML: A knowledge query and manipulation language. Technical report, Department of Computer Science, University of Maryland Baltimore County, 1992.

[15] Michael R. Genesereth. Designworld. In *Proceedings of the 1991 International Conference on Robotiocs and Automation*, pages 2785–2788, 1991.

[16] Michael R. Genesereth. Knowledge interchange format. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the Conference of the Principles of Knowledge Representation and Reasoning*, pages 599–600. Morgan Kaufmann Publishers, Inc., 1991.

[17] Michael R. Genesereth, Richard E. Fikes, and et al. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

[18] Thomas R. Gruber. The role of common ontology in achieving sharable, reusable knowledge bases. In J. A. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, pages 601–602, Cambridge, MA, 1991. Morgan Kaufmann.

[19] Thomas R. Gruber. Ontolingua: A mechanism to support portable ontologies. Technical Report KSL 91-66, Stanford University, Knowledge Systems Laboratory, 1992. June 1992 Revision.

[20] Michael N. Huhns, David M. Bridgeland, and Natraj V. Arni. A DAI communication aide. Technical Report ACT-RA-317-90, Microelectronics and Computer Technology Corporation, Microelectronics and Computer Technology Corporation, 3500 West Balcones Center Drive, Austin TX 78759-6509, October 1990.

[21] Nancy Lehrer. DARPA/Rolm Laboratory Planning and Scheduling Initiative, Knowledge Representation Specification Language: KRSL Version 2.0. Language specification and manual, 1992.

[22] Robert MacGregor. Loom users manual. Working Paper ISI/WP-22, USC/Information Sciences Institute, 1990.

[23] Don McKay, Tim Finin, and Anthony O'Hare. The intelligent database interface. In *Proceedings of the 7^{th} National Conference on Artificial Intelligence*, August 1990.

[24] Mark A. Musen. Overcoming the limitations of role-limiting methods. *Knowledge Acquisition*, 4(2):165–170, 1992.

[25] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2):235–249, May 1990.

[26] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In Sowa [34].

[27] Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):16–36, 1991.

[28] Mike P. Papazoglou and Timos K. Sellis. An organizational framework for cooperating intelligent information systems. *International Journal on Intelligent and Cooperative Information Systems*, 1(1), (to appear) 1992.

[29] J. Pastor, D. McKay, and T. Finin. Viewconcepts: Knowledge-based access to databases. In *Proceedings of the First International Conference on Information and Knowledgement*, November 1992.

[30] Peter F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39(2):263–272, June 1989.

[31] Christof Peltason, Albrecht Schmiedel, Carsten Kindermann, and Joachim Quantz. The BACK system revisited. KIT-Report 75, Department of Computer Science, Technische Universität Berlin, September 1989.

[32] Kirk Sayre and Michael A. Gray. Backtalk: A generalized dynamic communication system for DAI. Technical Report CSIS-91-004, The American University, Washington DC, August 1991.

[33] Sandip Sen and Edmund H. Durfee. A formal study of distributed meeting scheduling: preliminary results. In *Proceedings of the ACM Conference on Organizational Computing Systems*, pages 55–68, November 1991.

[34] John Sowa, editor. *Principles of Semantic Networks: Explorations in the representation of knowledge*. Morgan-Kaufmann, San Mateo, California, 1991.