KQML as an agent communication language *

Tim Finin, Yannis Labrou and James Mayfield
Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore MD USA
{finin,jklabrou,mayfield}@cs.umbc.edu

September, 1995

Abstract

One of the defining characteristics of an *information agent* is its ability for flexible interaction and interoperation with other, similar software agents. This focus on interoperability has been the foundation of the approach of the Knowledge Sharing Effort (KSE) in developing a basic framework for intelligent systems. We present KSE approach and the solutions suggested for the subproblems identified by the consortium, emphasizing the KSE's communication language and protocol KQML (Knowledge Query and Manipulation Language). In addition to presenting specific solutions, we are interested in demonstrating the conceptual decomposition of the problem of knowledge sharing into smaller more manageable problems, and in arguing that there is merit to those concepts independent of the success of individual solutions.

This paper will appear as a chapter in Jeff Bradshaw (Ed.), "Software Agents", MIT Press, Cambridge, to appear, (1995).

1 Introduction

It is doubtful that any conversation about agents will result in a consensus on the definition of an agent or of agency. From personal assistants and "smart" interfaces to powerful applications, and from autonomous, intelligent entities to information retrieval systems, anything might qualify as an agent these days. But, despite these different viewpoints, most imaginary conversants would agree that the ability for interaction and interoperation is desirable. The building block for intelligent interaction is knowledge sharing that includes both mutual understanding of knowledge and the communication of that knowledge. The importance of such communication is emphasized by Genesereth, who goes so far as to suggest that an entity is a software agent if and only if it communicates correctly in an agent communication language [10]. After all, it is hard to picture cyberspace with entities that exist only in isolation; it would go against our perception of a decentralized, interconnected electronic universe.

How might meaningful, constructive and intelligent interaction among software agents be provided? The same problem for humans requires more than the knowledge of a common language,

*This work was supported in part 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.

e.g., English; it also requires a common understanding of the terms used in a given context. A physicist's understanding of velocity is not the same as that of a car enthusiast,[1] and if the two want to converse about "fast" cars they have to speak a "common language." Also, humans must resort to a shared etiquette of communication, that is a result of societal development, and that is partially encoded in the language. Although we are not always conscious of doing so, we follow certain patterns when we ask questions or make requests. Such patterns have common elements across human languages. Likewise, for software agents to interact and interoperate effectively requires three fundamental and distinct components: 1) a common language; 2) a common understanding of the knowledge exchanged; and 3) the ability to exchange whatever is included in (1) and (2). We take effective interaction to be the exchange (communication) of information and knowledge that can be mutually understood. The more applications that can communicate with one another, and the wider the range of knowledge they can exchange, the better.

This perspective on interoperability in today's computing environment has been the foundation of the approach of the Knowledge Sharing Effort (KSE) consortium. We present the approach of the KSE and the solutions suggested for the subproblems identified by the consortium, emphasizing the KSE's communication language and protocol KQML (Knowledge Query and Manipulation Language). In addition to presenting specific solutions, we are interested in demonstrating the conceptual decomposition of the problem of knowledge sharing into smaller more manageable problems, and in arguing that there is merit to those concepts independent of the success of individual solutions.

In the remainder of this chapter we provide a brief coverage of the objectives and approach of the Knowledge Sharing Effort (Section 2), and a summary of the major results of the KSE (Section 3). In Section 4 we attempt to define the notion of a communication language and its desired features. In Section 5 we present the agent communication language KQML (a product of the KSE research) along with our notion of an environment of KQML-speaking agents. In Section 6 we evaluate KQML as a communication language. Section 7 provides an overview of applications and environments that have used KQML, and Section 8 surveys other communication languages (and/or approaches) with which we feel that KQML is (or should be) competing.

2 The approach of the Knowledge–Sharing Effort (KSE)

Let us address the issue of software agents and interoperability in more detail. We will refer to software agents as applications for which the ability to communicate with other applications and share knowledge is of primary importance. Figure 1 summarizes the possible components of such an agent; they are grouped into representation components, communication components, and components that are not directly related to shared understanding.

Mutual understanding of what is represented may be divided into two subproblems: 1) translation from one representation language to another (or from one family of representation languages to another); and 2) sharing the semantic content (and often the pragmatic content) of the represented knowledge among different applications. Translation alone is not sufficient because each knowledge base holds implicit assumptions about the meaning of what is represented. If two applications are to understand each other's knowledge, such assumptions must also be shared. That is, the semantic content of the various tokens must be preserved.

Communication is a threefold problem involving knowledge of (i) interaction protocol; (ii) com-

[1] Unless of course the physicist also happens to be interested in cars.

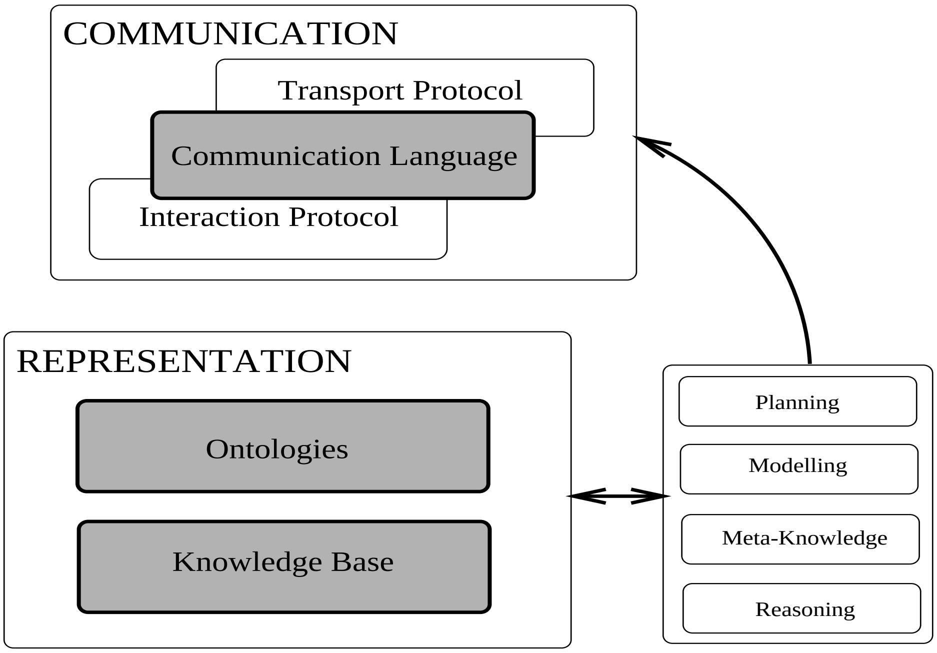

Figure 1: Our abstract model for interoperating software agents identifies three classes of components – representation components, communication components, and components not directly related to shared understanding.

munication language; and (iii) transport protocol. The interaction protocol refers to the high level strategy pursued by the software agent that governs its interaction with other agents. Such a protocol can range from negotiation schemes and game theory protocols to protocols as simple as "every time you do not know something, find someone who knows and ask." The communication language is the medium through which the attitudes regarding the content of the exchange are communicated. It is the communication language that suggests whether the content of the communication is an assertion, a request or some form of query. The transport protocol is the actual transport mechanism used for the communication, such as TCP, SMTP, HTTP, etc. Practical considerations may favor the use of one of those over others.[2]

Software agents may (or may not) have other components to help the agent carry out its business. The ability to reason about its own actions, to represent meta-knowledge, to plan activities or to model other agents can enhance the capabilities of an application. Such components are peripheral to the virtual knowledge base, although they are usually built on top of it. Although they may use the virtual knowledge base or the communication language to implement their agendas, the issues associated with them should be viewed as orthogonal to the issues of mutual understanding and communication.

The Knowledge–Sharing Effort (KSE), sponsored by the Advanced Research Projects Agency (ARPA), the Air Force Office of Scientific Research (ASOFR), the Corporation for National Research Initiative (NRI) and the National Science Foundation (NSF), is an initiative to develop technical infrastructure to support knowledge sharing among systems [18]. The KSE is organized around the following three working groups, each of which addresses a complementary problem identified incurrent knowledge representation technology: *Interlingua*, *Shared Reusable Knowledge*

[2]For example,for systems that use *firewalls* or are connected to a network only intermittantly,, SMTP may be preferred over TCP as a transport mechanism.

Bases, and *External Interfaces*.

The *Interlingua Group* is developing a common language for expressing the content of a knowledge–base. This group has published a specification document describing the *Knowledge Interchange Format*, or *KIF* [9]. KIF can be used to support translation from one content language to another, or as a common content language between two agents which use different native representation languages. Information about KIF and associated tools and is available over the Internet.[3] A group within the KSE with a similar scope is the *KRSS Group* (Knowledge Representation System Specification). This group focuses on the definition of common constructs within families of representation languages. The group has produced a common specification for terminological representations in the KL–ONE family.[4]

The *SRKB Group* (Shared, Reusable Knowledge Bases) is concerned with facilitating consensus on the content of sharable knowledge bases, with sub–interests in shared knowledge for particular topic areas and in topic–independent development tools and methodologies. It has established a repository for sharable ontologies and tools, which is available over the Internet.[5]

The scope of the *External Interfaces Group* is run–time interaction between knowledge–based systems and other modules in a run–time environment. Special attention has been given to two important cases—communication between two knowledge–based systems and communication between a knowledge–based system and a conventional database management system [21]. The KQML language is one of the main results to come out of the external interfaces group of the KSE.[6]

These three groups of the KSE roughly address the interoperability issues at the levels identified by the three shaded boxes of Figure 1; the results of the research efforts, namely KIF, Ontolingua and KQML, are the specific solutions suggested for them.

3 The solutions of the Knowledge–Sharing Effort

KIF is the solution suggested by the KSE for the syntactic aspects of representations for knowledge sharing. The language is intended as a powerful vehicle to express knowledge and meta–knowledge. There were two different motivations behind the development of a language like KIF: 1) creation of a *lingua franca* for the development of intelligent applications, with an emphasis on interoperation (in cooperation with the other components of the "package solution" of the KSE); and 2) creation of a common interchange format so that with the use of "translators" one could translate from language A to KIF and from KIF to language B instead of translating from A to B.[7] KIF has found its way into applications,[8] but it remains to be proven whether it will fulfill any of its intended roles.

Next we provide a brief coverage of *KIF*, the ideas behind ontologies and *Ontolingua* (the framework for the development of ontologies) and KQML. This is not intended as a detailed analysis, but rather as an introduction to the main ideas of these research efforts. The remainder of this

[3]The URL is `http://www.cs.umbc.edu/kse/kif/`.

[4]This document and other information on the KRSS group is available as `http://www.cs.umbc.edu/kse/-krss/`.

[5]The URL is `http://www.cs.umbc.edu/kse/srkb/`.

[6]General information about KQML is available from `http://www.cs.umbc.edu/kqml/`.

[7]So for n languages the number of translators needed would be $2n$ instead of $n*(n-1)$. Of course the advantages of having a common interchange format go beyond such a reduction; but defining the advantages (and disadvantages) of such an approach has been an issue of debate.

[8]Primarily in conjunction with ACL, an implementation of KQML, which is standard KQML with a commitment to KIF as the content language.

presentation is primarily concerned with KQML and its function as a communication language for software agents.

3.1 Knowledge Interchange Format (KIF)

KIF[9] is a prefix version of first order predicate calculus with extensions to support non–monotonic reasoning and definitions. The language description includes both a specification for its syntax and one for its semantics. First and foremost, KIF provides for the expression of simple data. For example, the sentences shown below encode three-tuples in a personnel database in which the fields represent employee ID number, department assignment and salary, respectively.

```
(salary 015-46-3946 widgets 72000)
(salary 026-40-9152 grommets 36000)
(salary 415-32-4707 fidgets 42000)
```

More complicated information can be expressed through the use of complex terms. For example, the following sentence states that one chip is larger than another:

```
(> (* (width chip1) (length chip1))
   (* (width chip2) (length chip2)))
```

KIF includes a variety of logical operators to assist in the encoding of logical information (such as negation, disjunction, rules, quantified formulas, and so forth). The expression shown below is an example of a complex sentence in KIF. It asserts that the number obtained by raising any real–number `?x`[10] to an even power `?n` is positive:

```
(=> (and (real-number ?x) (even-number ?n))
    (> (expt ?x ?n) 0))
```

KIF provides for the encoding of knowledge about knowledge, using the backquote (`‘`) and comma (`,`) operators and related vocabulary. For example, the following sentence asserts that agent Joe is interested in receiving triples in the salary relation.[11]

```
(interested joe ‘(salary ,?x ,?y ,?z))
```

KIF can also be used to describe procedures, *i.e.* to write programs or scripts for agents to follow. Given the prefix syntax of KIF, such programs resemble Lisp or Scheme. The following is an example of a three–step procedure written in KIF. The first step ensures that there is a fresh line on the standard output stream; the second step prints `Hello!` to the standard output stream; the final step adds a carriage return to the output.

```
(progn (fresh-line t)
       (print "Hello!")
       (fresh-line t))
```

[9]This presentation of KIF is based on a similar presentation [10].

[10]Symbols beginning with a question mark are taken to be variables.

[11]The use of commas signals that the variables should not be taken literally. Without the commas, this sentence would say that agent joe is interested in the sentence `(salary ?x ?y ?z)` instead of its instances.

The semantics of the KIF core (KIF without rules and definitions) is similar to that of first order logic. There is an extension to handle nonstandard operators (like ‘ and ,), and there is a restriction that models satisfy various axiom schemata (to give meaning to the basic vocabulary in the format). Despite these extensions and restrictions, the core language retains the fundamental characteristics of first order logic, including compactness and the semi–decidability of logical entailment.

3.2 Ontologies and Ontolingua

The SRKB Working Group is also working on the problem of sharing the content of formally represented knowledge. Sharing content requires more than a formalism (such as KIF) and a communication language (KQML). Although the problem of understanding what must be held in common among communicating agents is a fundamental question of philosophy and science, the SRKB is focusing on the practical problem of building knowledge–based software that can be reused as off–the–shelf technology. The approach is to focus on common ontologies [18]. Every knowledge–based system relies on some conceptualization of the world (objects, qualities, distinctions and relationships that matter for performing some task) that is embodied in concepts, distinctions, *etc.* in a formal representation scheme. A **common ontology** refers to an explicit specification of the ontological commitments of a set of programs. Such a specification should be an objective (*i.e.*, interpretable outside of the program) description of the concepts and relationships that the programs use to interact with other programs, knowledge bases, and human users. An agent commits to an ontology if its observable actions are consistent with the definitions in the ontology.

The SRKB Group has worked on the construction of ontologies for various domains. Ontologies are written in KIF, using the definitional vocabulary of Ontolingua.[12] Each ontology defines a set of classes, functions, and object constants for some domain of discourse, and includes an axiomatization to constrain the interpretation. The resulting language (the basic logic of KIF + the vocabulary and theory from the ontologies) allows for the sentences to be interpreted unambiguously and independent of context, making the relevant detail explicit. These ontologies can then be used by communicating application. There has been a considerable number of ontologies developed by the group, on a variety of domains that might be of interest to software applications.

3.3 Knowledge Query Manipulation Language (KQML)

KQML was conceived as both a message format and a message–handling protocol to support run–time knowledge sharing among agents. The key features of KQML may be summarized as follows:

- KQML messages are opaque to the content they carry. KQML messages do not merely communicate sentences in some language, but they rather communicate an attitude about the content (assertion, request, query).

- The language's primitives are called *performatives*. As the term suggests, the concept is related to speech act theory. Performatives define the permissible actions (operations) that agents may attempt in communicating with each other.

- An environment of KQML speaking agents may be enriched with special agents, called *facilitators*, that provide such functions as: association of physical addresses with symbolic names; registration of databases and/or services offered and sought by agents; and communication

[12] See http://www-ksl.stanford.edu/knowledge-sharing/ontologies/README.html for more information on ontologies and the various projects with which the SRKB Group has been involved.

services (forwarding, brokering *etc.*). To use a metaphor, facilitators act as efficient secretaries for the agents in their domain.

Intelligent interaction is more than an exchange of messages. As suggested in Section 2, KQML is an attempt to dissociate these issues from the communication language, which should define a set of standard message types that are to be interpreted identically by all interacting parties. A *universal* communication language is of interest to a wide range of applications that need to communicate something more than pre–defined or fixed statements of facts. KQML is the centerpiece of this presentation and is described in more detail in Section 5.

4 Communication languages and their desired features

It is fair to ask whether one can talk about an agent communication language without referring to the properties of agency. Instead of providing a comprehensive definition of agency we suggest that agents are commonly taken to be "high–level" (*i.e.*, they use symbolic representation, display cognitive–like function, and/or have a belief and/or a knowledge store), and are commonly viewed as having an intentional level description (*i.e.*, their state is viewed as consisting of mental components such as beliefs, capabilities, choices, commitments *etc.*). We take this "description" to be a helpful *abstract model* for *viewing* software agents, even if their actual implementation does not make claims to such ambitious concepts. Agents then reside at the *knowledge level* [19, 20] and are not well served by general languages and protocols developed for distributed computing. Such languages and protocols focus on *processes* rather than on the programs or collection of programs that constitute the agents. As a result, a communication language should be powerful enough to support communication between programs that are viewed as being at this higher level (with an intentional description); otherwise the agents will have to bear the task of translating between the lower level and the agent's level.

It should also be made clear that a communication language is not a protocol, although both may be concerned with communication and communication-related issues. The distinction between a communication language and a protocol is often fuzzy. A protocol, as used or mentioned in the context of communication languages, may have any of the following three meanings: (i) a transport protocol, like HTTP, SMTP, FTP, etc.; (ii) a high level framework for interaction, such as negotiation, game theory protocols, planning, *etc.*; or (iii) constraints on the possible valid exchanges of communication primitives.[13] A communication language may use protocols of the first kind as transport mechanisms, may be used by protocols of the second kind as a way to implement them, and usually includes protocols of the third kind as part of its description; but a communication language definitely is not merely a protocol itself.

In this section we suggest requirements for agent communication languages. We divide these requirements into seven categories: form, content, semantics, implementation, networking, environment, and reliability. We believe that an agent communication language will be valuable to the extent that it meets these requirements. At times, these requirements may be in conflict with one another. For example, a language that can be easily read by people might not be as concise as possible. It is the job of the language designer to balance these various needs. In Section 6 we evaluate KQML with respect to these requirements.

[13] Very much as in meaningful communications between sane human beings, where a question about the time for example will be followed by a response (hopefully about time) and not by another question about the weather.

Form. A good agent communication language should be declarative, syntactically simple, and readable by people. It should be concise, yet easy to parse and to generate. To transmit a statement of the language to another agent, the statement must pass through the bit stream of the underlying transport mechanism. Thus, the language should be linear or should be easily translated into a linear form. Finally, because a communication language will be integrated into a wide variety of systems, its syntax should be extensible.

Content. A communication language should be layered in a way that fits well with other systems. In particular, a distinction should be made between the communication language, which expresses communicative acts, and the content language, which expresses facts about the domain. Such layering facilitates the successful integration of the language to applications while providing a conceptual framework for the understanding of the language. The language should commit to a well defined set of communicative acts (primitives). Although this set could be extensible, a core of primitives that capture most of our intuitions about what constitutes a communicative act irrespective of application (database, object-oriented system, knowledge base, *etc.*) will ensure the usability of the language by a variety of systems. The choice of the core set of primitives also relates to the decision of whether to commit to a specific content language. A commitment to a content language allows for a more restricted set of communicative acts because it is then possible to carry more information at the content language level. The disadvantage is that all applications must then use the same content language; this is a heavy constraint.

Semantics. Although the semantic description of communication languages and their primitives is often limited to natural language descriptions, a more formal description is necessary if the communication language is intended for interaction among a diverse range of applications. Applications designers should have a shared understanding of the language, its primitives and the protocols associated with their use, and abide by that shared understanding. The semantics of a communication language should exhibit those properties expected of the semantics of any other language. It should be grounded in theory, and it should be unambiguous. It should exhibit canonical form (similarity in meaning should lead to similarity in representation). Because a communication language is intended for interaction that extends over time amongst spatially dispersed applications, location and time should be carefully addressed by the semantics. Finally, the semantic description should provide a model of communication, which would be useful for performance modeling, among other things.

Implementation. The implementation should be efficient, both in speed and in bandwidth utilization. It should provide a good fit with existing software technology. The interface should be easy to use; details of the networking layers that lie below the primitive communicative acts should be hidden from the user. Finally, the language should be amenable to partial implementation, because simple agents may only need to handle a subset of the primitive communicative acts.

Networking. An agent communication language should fit well with modern networking technology. This is particularly important because some of the communication will be *about* concepts involving networked communications. The language should support all of the basic connections—point-to-point, multicast and broadcast. Both synchronous and asynchronous connections should be supported. The language should contain a rich enough set of primitives that it can serve as a substrate upon which higher-level languages and interaction protocols can be built. Moreover, these higher-level protocols should be independent of the transport mechanisms (*e.g.*, TCP/IP, email, http, *etc.*) used.

Environment. The environment in which intelligent agents will be required to work will be highly distributed, heterogeneous, and extremely dynamic. To provide a communication channel to the outside world in such an environment, a communication language must provide tools for coping with heterogeneity and dynamism. It must support interoperability with other languages and protocols. It must support knowledge discovery in large networks. Finally, it must be easily attachable to legacy systems.

Reliability. A communication language must support reliable and secure communication among agents. Provisions for secure and private exchanges between two agents should be supported. There should be a way to guarantee authentication of agents. We should not assume that agents are infallible or perfect—they should be robust to inappropriate or malformed messages. The language should support reasonable mechanisms for identifying and signaling errors and warnings.

5 Knowledge Query Manipulation Language (KQML)

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 [9], extended SQL, and LOOM [16] have their supporters, but there is also a strong position that it is too early to standardize on any representation language [13]. 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 in order 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 communicatinf software agewnts 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. 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 such agents. In the next two sections we present the KQML language, its primitives and protocols supported, and the software environment of KQML–speaking applications.

5.1 A description of KQML

The KQML language can be thought of as consisting 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 programs 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, very 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 which describe the lower level communication parameters, such as the identity of the sender and recipient, and a unique identifier associated with the communication.

It is the message layer that 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 which 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 beopaque to a KQML-speaking agent, this layer also includes optional features which describe the content language, the ontology it assumes, and some type of 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 even though their content is inaccessible.

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 initial implementations, which were done in Common Lisp; it has turned out 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 is the content level, the values of the `:reply-with`, `:sender`, `:receiver` keywords form the communication layer and the performative name, with the `:language` and `:ontology` form the message layer. In due time, *stock-server* might send to *joe* the following KQML message:

```
(tell
  :sender stock-server
  :content (PRICE IBM 14)
  :receiver joe
  :in-reply-to ibm-stock
  :language LPROLOG
  :ontology NYSE-TICKS)
```

A query similar to the *ask-all* 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,Time))"
  :receiver stock-server
  :language standard_prolog
  :ontology NYSE-TICKS)
```

The first message asks for a single reply; the second asks for a set as a reply. If we had posed a query which had a large number of replies, would could ask that they each be sent separately, instead of as a single large collection by changing the performative. (To save space, we will no longer repeat fields which 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 set of replies. To exert control of 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 language content and it requests that the agent receiving the request take the stream of messages and hold them and release them 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 sending agent sends either 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 is a stream of messages which are generated as the trading price of IBM stock changes.

Though there is a predefined 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 Table 1. In addition to standard communication performatives such as *ask*, *tell*, *deny*, *delete*, and more protocol oriented performatives such as *subscribe*, KQML contains performatives related to the non–protocol aspects of pragmatics, such as *advertise*—which allows an agent to announce what kinds of KQML messages it is willing to handle; and *recruit*—which can be used to find suitable agents for particular types of messages (the uses of these performatives are described in the next section). For example, the server in the above example might have earlier announced:

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,

Table 1: There are about two dozen reserved performative names which fall into seven basic categories.

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

Which 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.

There are a variety of interprocess information exchange protocols in KQML. In the simplest, one agent acts as a client and sends a query to another agent acting as a server and then waits for a reply, as is shown between agents A and B in Figure 2. The server's reply might consist of a single answer or a collection or set of answers. In another common case, shown between agents A and C, the server's reply is not the complete answer but a handle which allows the client to ask for the components of the reply, one at a time. A common example of this exchange occurs when a client queries a relational database or a reasoner which produces a sequence of instantiations in response. Although this exchange requires that the server maintain some internal state, the individual transactions are as before—involving a *synchronous* communication between the agents. A somewhat different case occurs when the client subscribes to a server's output and an indefinite number of *asynchronous* replies arrive at irregular intervals, as between agents A and D in Figure 2. The client does not know when each reply message will be arriving and may be busy performing some other task when they do.

There are other variations of these protocols. Messages might not be addressed to specific hosts,

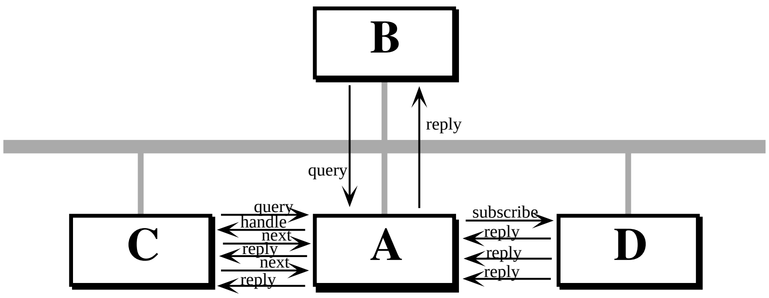

Figure 2: Several basic communication protocols are supported in KQML.

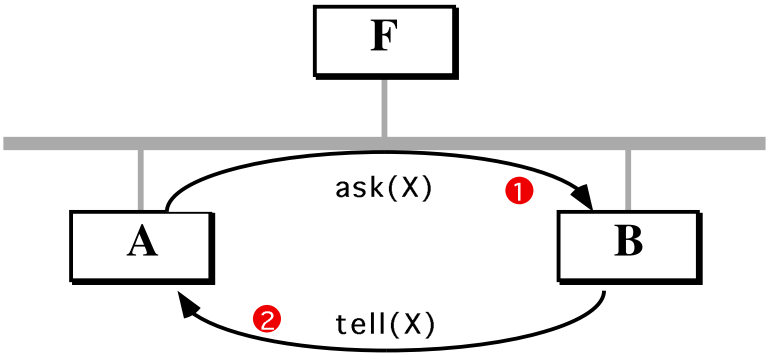

Figure 3: When A is aware of B and of the appropriateness of querying B about X, a simple point-to-point protocol can be used.

but broadcast to a number of them. The replies, arriving synchronously or asynchronously have to be collated and, optionally, associated with the query that they are replying to.

5.2 Facilitators, mediators and the environment of KQML-speaking agents

One of the design criteria for KQML was to produce a language that could support a wide variety of interesting agent architectures. Our approach to this is to introduce a small number of KQML performatives which are used by agents to describe the meta-data specifying the information requirements and capabilities and then to introduce a special class of agents called *communication facilitators* [10]. A facilitator is an agent that performs various useful communication services, *e.g.* maintaining a registry of service names, forwarding messages to named services, routing messages based on content, providing "matchmaking" between information providers and clients, and providing mediation and translation services.

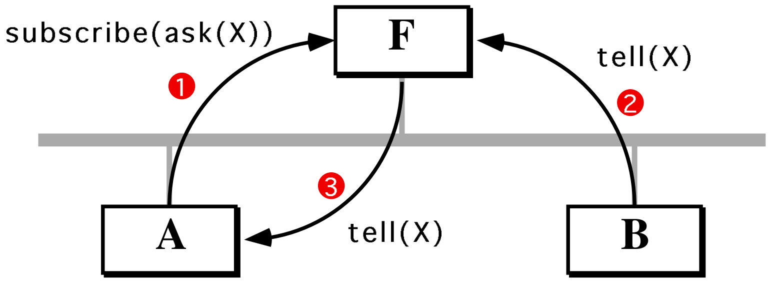

Figure 4: Agent A can ask facilitator agent F to monitor for changes in its knowledge-base. Facilitators are agents that deal in knowledge about the information services and requirements of other agents and offer such services as forwarding, brokering, recruiting and content-based routing.

As an example, consider a case where an agent A would like to know the truth of a sentence X, and agent B may have X in its knowledge-base, and a facilitator agent F is available. If A is aware that it is appropriate to send a query about X to B, then it can use a simple *point to point* protocol and send the query directly to B, as in Figure 3.

If, however, A is not aware of what agents are available, or which may have X in their knowledge-bases, or how to contact those of whom it is aware, then a variety of approaches can be used. Figure 4 shows an example in which A uses the *subscribe* performative to request that facilitator F monitor for the truth of X. If B subsequently informs F that it believes X to be true, then F can in turn inform A.

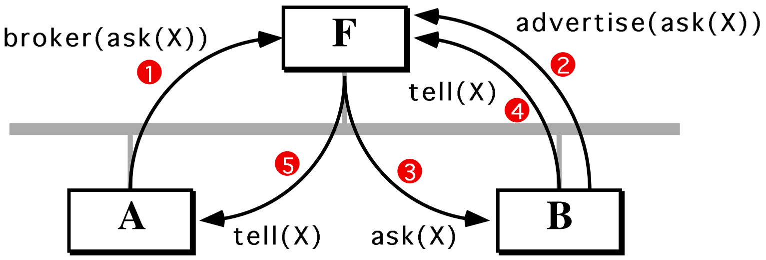

Figure 5: The broker performative is used to ask a facilitator agent to find another agent which can process a given performative and to receive and forward the reply.

Figure 5 shows a slightly different situation. A asks F to find an agent that can process an *ask(X)* performative. B independently informs F that it is willing to accept performatives matching *ask(X)*. Once F has both of these messages, it sends B the query, gets a response and forwards it to A.

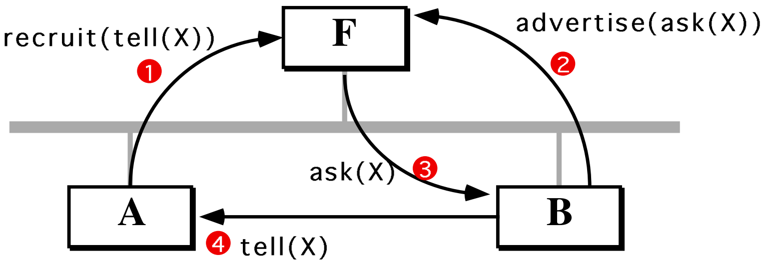

Figure 6: The recruit performative is used to ask a facilitator agent to find an appropriate agent to which an embedded performative can be forwarded. Any reply is returned directly to the original agent.

Figure 6, A uses a slightly different performative to inform F of its interest in knowing the truth of X. The recruit performative asks the recipient to find an agent that is willing to receive and process an embedded performative. That agent's response is then to be directly sent to the initiating agent. Although the difference between the examples used in Figures 5 and 6 are small for a simple ask query, consider what would happen if the embedded performative was *subscribe(ask-all(X))*.

As a final example, consider the exchange in Figure 7 in which A asks F to "recommend" an agent to whom it would be appropriate to send the performative *ask(X)*. Once F learns that B is willing to accept *ask(X)* performatives, it replies to A with the name of agent B. A is then free to initiate a dialogue with B to answer this and similar queries.

From these examples, we can see that one of the main functions of facilitator agents is to help other agents find appropriate clients and servers. The problem of how agents find facilitators in the first place is not strictly an issue for KQML and has a variety of possible solutions.

Current KQML-based applications have used one of two simple techniques. In the PACT project [6], for example, all agents used a central, common facilitator whose location was a parameter initialized when the agents were launched. In the ARPI applications [4], finding and establishing contact with a local facilitator is one of the functions of the KQML API. When each agent starts up,

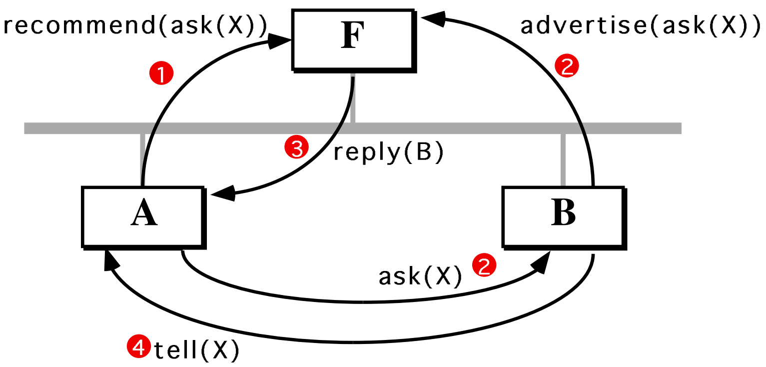

Figure 7: The recommend performative is used to ask a facilitator agent to respond with the "name" of another agent which is appropriate for sending a particular performative.

its KQML router module announces itself to the local facilitator so that it is registered in the local database. When the application exits, the router sends another KQML message to the facilitator, removing the application from the facilitator's database. By convention, a facilitator agent should be running on a host machine with the symbolic address *facilitator.domain* and listening to the standard KQML port.

6 Evaluation of KQML as an agent communication language

In this section, we evaluate the KQML language as it stands today, relative to our requirements for agent communication languages, given in Section 4.

Form. The only primitives of the languages, *i.e.*, the performatives, convey the communicative act and the actions to be taken as a result. Thus the form should be deemed to be declarative. In format, KQML messages are linear streams of characters with a Lisp-like syntax. Although this formatting is irrelevant to the functions of the language, it makes the messages easy to read, to parse and to convert to other formats.[14] The syntax is simple and allows for the addition of new parameters, if deemed necessary, in future revisions of the language.

Content. The KQML language can be viewed as being divided into three layers: the content layer, the message layer and the communication layer. KQML messages are oblivious to the content they carry. Although in current implementations of the language there is no support for non–ASCII content, there is nothing in the language that would prevent such support. The language offers a minimum set of performatives that covers a basic repertoire of communicative acts. They constitute the message layer of the language and are to be interpreted as speech acts. Although there is no "right" necessary and sufficient set of communicative acts, KQML designers tried to find the middle ground between the two extremes: 1) providing a small set of primitives thereby requiring overloading at the content level; and 2) providing an extensive set of acts, where inevitably acts will overlap one another and/or embody fine distinctions. The *communication layer* encodes a set of features to the message which describe the lower level communication parameters, such as the identity of the sender and recipient, and a unique identifier associated with the communication.

[14]Simple programs exist to convert KQML messages to predicate-like format.

Semantics. KQML semantics is still an open issue. For now there are only natural language descriptions of the intended meaning of the performatives and their use (protocols). An approach that emphasizes the speech act flavor of the communication acts is a thread of ongoing research [14].

Implementation. In two current implementations of KQML interfaces, the Lockheed KQML API and the Loral/UMBC KQML API, each provides a content–independent message router and a facilitator. The application must provide handler functions for the performatives in order for the communication acts to be processed by the application and eventually return the proper response(s). It is not necessary that an application should handle all performatives since not all KQML-speaking applications will be equally powerful. Creating a KQML speaking interface to an existing application is a matter of providing the handler functions. The efficiency of KQML communication has been investigated. Various compression enhancements have been added which cut communication costs by reducing message sizes and also by eliminating a substantial fraction of symbol lookup and string duplication.

Networking. KQML–speaking agents can communicate directly with other agents (addressing them by symbolic name), broadcast their messages or solicit the services of fellow agents or facilitators for the delivery of a message by using the appropriate performatives (see Section 7). KQML allows for both synchronous/asynchronous interactions and blocking/non-blocking message sending on behalf of an application through assignment of the appropriate values for those parameters in a KQML message.

Environment. KQML can use any transport protocol as its transport mechanism (HTTP, SMTP, TCP/IP, etc.). Also, because KQML messages are oblivious to content, there are no restrictions on the content language beyond the provision of functions that handle the performatives for the content language of the application. Interoperability with other communication languages remains to be addressed as such languages appear. One such attempt has been made by Davis, whose Agent–K attempts to bridge KQML and Shoham's Agent Oriented Programming [22]. The existence of facilitators in the KQML environment can provide the means for knowledge discovery in large networks, especially if facilitators can cooperate with other knowledge discovery applications available in the World Wide Web.

Reliability. The issues of security and authentication have not been addressed properly thus far by the KQML community. No decision has been made on whether they should be handled at the transport protocol level or at the language level. At the language level, new performatives or message parameters can be introduced that allow for encryption of either the content or the whole KQML message. Since KQML speaking agents might be imperfect, there are performatives (such as *error* and *sorry*) that can be used as responses to messages that an application cannot process or comprehend.

7 Applications of KQML

The KQML language and implementations of the protocol have been used in several prototype and demonstration systems. The applications have ranged from concurrent design and engineering of hardware and software systems, military transportation logistics planning and scheduling, flexible architectures for large–scale heterogeneous information systems, agent–based software integration and cooperative information access planning and retrieval. KQML has the potential to significantly enhance the capabilities and functionality of large–scale integration and interoperability efforts now underway in communication and information technology such as the national information infrastructure and OMG's CORBA, as well as in application areas in electronic commerce, health information systems and virtual enterprise integration. The content languages used have included languages intended for knowledge exchange including the Knowledge Interchange Format (KIF) and the Knowledge Representation Specification Language (KRSL) [15] as well as other more traditional languages such as SQL. Early experimentations with KQML began in 1990. The following is a representative selection of applications and experiments developed using KQML.

The design and engineering of complex computer systems, whether exclusively hardware or software systems or both, today involves multiple design and engineering disciplines. Many such systems are developed in fast cycle or concurrent processes which involve the immediate and continual consideration of end–product constraints, e.g., marketability, manufacturing planning, etc. Further, the design, engineering and manufacturing components are also likely to be distributed across organizational and company boundaries. KQML has proved highly effective in the integration of diverse tools and systems enabling new tool interactions and supporting a high–level communication infrastructure reducing integration cost as well as flexible communication across multiple networking systems. The use of KQML in these demonstrations has allowed the integrators to focus on what the integration of design and engineering tools can accomplish and appropriately deemphasized how the tools communicate [11, 17, 7, 8].

KQML has been used as the communication language in several technology integration experiments in the ARPA Rome Lab Planning Initiative. One of these experiments supported an integrated planning and scheduling system for military transportation logistics linking a planning agent (in SIPE [23, 3]), with a scheduler (in Common Lisp), a knowledge base (in LOOM [16]), and a case based reasoning tool (in Common Lisp). All of the components integrated were pre-existing systems which were not designed to work in a cooperative distributed environment. In a second experiment, an information agent was constructed by integrating CoBASE [5], a cooperative front–end, SIMS [1, 2], an information mediator for planning information access, and LIM [21], an information mediator for translating relational data into knowledge structures. CoBASE processes a query, and, if no responses are found relaxes the query based upon approximation operators and domain semantics and executes the query again. CoBASE generates a single knowledge–based query for SIMS which uses knowledge of different information sources to select which of several information sources to access, partitions the query and optimizes access. Each of the resulting queries in this experiment is sent to a LIM knowledge server which answers the query by creating objects from tuples in a relational database. A LIM server front–ends each different database. This experiment was run over the internet involving three, geographically dispersed sites.

Agent–Based Software Integration [12] is an effort underway at Stanford University which is applying KQML as an integrating framework for general software systems. Using KQML, a federated architecture incorporating a sophisticated facilitator is developed which supports an agent–based view of software integration and interoperation [10]. The facilitator in this architecture is an intelligent agent which processes and reasons about the content of KQML messages, supporting tighter integration of disparate software systems.

Other work done at Stanford involves ACL (Agent Communication Language), an implementation of KQML that differs from "pure" KQML in the commitment it makes to KIF as the content language of the interacting applications. ACL has been used in several large–scale demonstrations of software interoperation, and the results are promising. Full specifications are available, and parts of the language are making their way through various standards organizations. Several start–up companies are proposing to offer commercial products for processing ACL; and a number of established computer system vendors are looking at ACL as a possible language for communication among heterogeneous systems. Genesereth provides more about the specifics of this approach [10] whose success is tied to the advantages and feature of KIF.

We have also successfully used KQML in other smaller demonstrations integrating distributed clients (in C) with mediators which were retrieving data from distributed databases. Mediators are not just limited distributed database access. In another demonstration, we experimented with a KQML URL for the World Wide Web. The static nature of links within such hypermedia structures lends itself to be extended with virtual and dynamic links to arbitrary information sources as can be supported easily with KQML.

8 Other communication languages

There has not been much work on communication languages from a practitioner's point of view. If we set aside work on network (transport) protocols or protocols in distributed computing as being too inefficient for the purposes of intelligent agents (as opposed to processes) the rest of the relevant research may be divided into *theoretical* and the *applied* approaches. The more theoretical work offers constructs and formalisms which address issues of agency in general and communication in particular, as a dimension of agent behavior, with an emphasis on the intentional description of an agent;[15]. The more applied work is aimed at developing agent languages and associated communication protocols focused on tools, interfaces and supporting applications. To our knowledge, there has not been any work on communication languages that is part a broader project that commits to specific architectures.

Agent Oriented Programming

Although Agent Oriented Programming (AOP) could be classified in the first one of the categories mentioned above, the fact that it comes with a programming language in which one can program agents that communicate and evolve, makes it more than a construct of a theoretical interest. In AOP [22] agents are viewed as entities whose state is viewed as consisting of mental components such as beliefs, capabilities, choices and commitments. Although this so called *intentional* description of a software system is nothing new, AOP introduces a formal language with syntax and semantics to describe the mental states and an interpreted programming language, called AGENT–0 that has semantics consistent with those of the mental states, and in which a programmer can program an agent. A part of AGENT–0, [16] is a communication language that introduces primitives for the interaction of agents. The primitives are speech acts, their semantics are provided in terms of their execution (the communication acts update the belief and the commitment space of an agent), and may be executed conditionally (according to whether certain mental states hold).

[15] An agent theory is concerned with how an agent's knowledge, actions and cognitive state relate to one another, guide the agent's "behavior" and affect both the agent and the environment in which the agent finds itself, through time.

[16] AGENT–O is only one of the possible programming languages in the AOP paradigm. In AOP there is no "proper" programming language. Extensions or replacements of AGENT–O may be introduced but they will have to be consistent with the intentional description introduces by the paradigm.

Since AGENT–O was intended as a prototype to illustrate the principles of AOP, it is limited in many important ways. For example, facts have to be atomic sentences (i.e., no conjunction, disjunction, or modal operators allowed) and commitments can only be made for primitive actions (i.e., no planning). An attempt to alleviate some of the deficiencies of AGENT–0 was done with PLACA by including operators for planning to do actions and achieve goals, but PLACA is also an experimental language and has not reached production state. AGENT–K[17] is a language in the AOP that essentially extends AGENT–0 to use KQML for communication. This still retains the issue of cooperation of agents written in the AOP paradigm with agents that do not fall in the AOP paradigm. Finally it remains to be proven the extent to which the AOP paradigm provides a powerful enough framework for serious agent programming.

Telescript

Telescript, a product of *General Magic*, defines an environment for transactions between software applications over the network, with a focus on applications in the area of electronic commerce. In the Telescript paradigm agents "travel" over the network (carrying both procedures and data) and perform actions on data at the transport location, instead of exchanging the data. Its developers suggest that this approach offers advantages on issues of bandwidth use and security, with respect to the predominant client–server paradigm. Telescript is an interpreted, communication–centric language executed by the Telescript engine that has access to the application environment through an API. A typical communicating application is written partly in Telescript and partly in some other (host) language with Telescript having control of all communication related issues, such as transporting the agent, handling conditions, scheduling the agents activities, gathering and modifying information, etc. Telescript has attracted the interest of commercial vendors for electronic commerce applications.

Can Telescript agents interoperate with other (non–Telescript) agents? First of all, Telescript agents do not communicate, they transport themselves on location and execute a pre–defined script. Even if some form of communication was allowed it is unknown: 1) how communication would be integrated to this new paradigm of transportable, script–executing paradigm of agents (will agents exchange scripts?), and 2) how can Telescript agents interoperate with agents whose interaction follow the traditional client–server paradigm. Although such questions have not been addressed yet it seems that Telescript agents will be confined to a Telescript universe.

9 Conclusions

There is no silver bullet for the problem of knowledge sharing. The difficulties of addressing this issue go beyond the plethora of applications and systems that would be candidates for knowledge sharing. The problem itself is not a *single*, *well–defined* problem but rather a wide range of subproblems (and corresponding approaches). From this point of view we feel that the KSE approach and KQML have merits that go beyond the success of the individual solutions that have been suggested. Above all the KSE promotes an approach to the problem and the fundamental subproblems to be addressed. This approach includes: a) translation between representations; b) sharing the semantic (and often pragmatic) content of the knowledge that is represented; and c) communicating attitudes about the shared knowledge. Whether the respective solutions for these three problems

[17] Look at URL http://www.csd.abdn.ac.uk/ pedwards/publs/agentk.html for more details.

will be KIF, Ontolingua and KQML is to be proven in time, but we believe that these are the three levels at which the overall problem should be attacked.

We believe the same argument to be true for KQML. The communication language should be dissociated from interaction and transport protocols, should be oblivious to the content and concerned only with attitudes about the content. The idea of a communication language that offers primitives (the performatives), modeled after speech acts, that have a meaning outside the context of a specific application or representation is not necessarily a new one, but KQML is a first attempt for a communication language based on this concept. The number and the variety of the primitives will always be a matter of debate; KQML developers tried to balance the two extremes of having very few primitives (and thus overloading the content) and offering an extensive set of performatives that would inevitably overlap and would be hard to standardize. The other idea offered by KQML is that of having specialized agents, called *facilitators*, that with the use of the appropriate KQML performatives can help agents find other agents (or be found by other agents) that can perform desired tasks for them.

References

[1] Yigal Arens. Planning and reformulating queries for semantically-modeled multidatabase systems. In *First International Conference on Information and Knowledge Management*, October 1992.

[2] Yigal Arens, Chin Chee, Chun-Nan Hsu, Hoh In, and Craig A. Knoblock. Query processing in an information mediator. In *Proceedings of the ARPA/Rome Lab 1994 Knowledge-Based Planning and Scheduling Initiative Workshop*, February 1994.

[3] Marie Bienkowski, Marie desJardins, and Roberto Desimone. SOCAP: system for operations crisis action planning. In *Proceedings of the ARPA/Rome Lab 1994 Knowledge-Based Planning and Scheduling Initiative Workshop*, February 1994.

[4] Mark Burstein, editor. *Proceedings of the ARPA/Rome Lab 1994 Knowledge-Based Planning and Scheduling Initiative Workshop.* Morgan Kuafmann Publishers, Inc., February 1994.

[5] Wes Chu and Hua Yang. Cobase: A cooperative query answering system for database systems. In *Proceedings of the ARPA/Rome Lab 1994 Knowledge-Based Planning and Scheduling Initiative Workshop*, February 1994.

[6] M. Cutkosky, E. Engelmore, R. Fikes, T. Gruber, M. Genesereth, and W. Mark. PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer*, pages 28–38, January 1993.

[7] D. Kuokka et. al. Shade: Technology for knowledge-based collaborative. In *AAAI Workshop on AI in Collaborative Design*, 1993.

[8] William Mark et. al. Cosmos: A system for supporting design negotiation. *Journal of Concurrent Engineering: Applications and Research (CERA)*, 2(3), 1994.

[9] M. Genesereth and R. Fikes et. al. Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, 1992.

[10] Michael R. Genesereth and Steven P. Katchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 147, 1994.

[11] Mike Genesereth. Designworld. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 2,785–2,788. IEEE CS Press.

[12] Mike Genesereth. An agent-based approach to software interoperability. Technical Report Logic-91-6, Logic Group, CSD, Stanford University, February 1993.

[13] Matt Ginsberg. Knowledge interchange format: The KIF of death. *AI Magazine*, 1991.

[14] 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 as `http://www.cs.umbc.edu/kqml/-papers/kqml-semantics.ps`.

[15] Nancy Lehrer. The knowledge representation specification language manual. Technical report, ISX Corporation, Thousand Oaks, California, 1994.

[16] Robert MacGregor and Raymond Bates. The LOOM knowledge representation language. Technical Report ISI/RS-87-188, USC/ISI, 1987. Also appears in *Proceedings of the Knowledge-Based Systems Workshop* held in St. Louis, Missouri, April 21–23, 1987.

[17] M.Tenenbaum, J. Weber, and T. Gruber. Enterprise integration: Lessons from shade and pact. In C. Petrie, editor, *Enterprise Integration Modeling*. MIT Press, 1993.

[18] 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.

[19] Allen Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[20] Allen Newell. Reflections on the knowledge level. *Artificial Intelligence*, 59:31–38, 1993.

[21] Jon Pastor, Don McKay, and Tim Finin. View-concepts: Knowledge-based access to databases. In *First International Conference on Information and Knowledge Management*, October 1992.

[22] Yoav Shoham. Agent–oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[23] David Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers, Inc., San Mateo, CA., 1988.