Desiderata for Agent Communication Languages *

James Mayfield **Yannis Labrou** **Tim Finin**

Computer Science Department
University of Maryland Baltimore County
Baltimore MD 21228–5398 USA
{mayfield,jklabrou,finin}@cs.umbc.edu

Abstract

This paper offers some opinions on the desirable features of languages and protocols for communication among intelligent information agents. These desiderata are divided into seven categories: form, content, semantics, implementation, networking, environment, and reliability. The Knowledge Query and Manipulation Language (KQML), is a new language and protocol for exchanging information and knowledge. This work is part of a larger effort, the ARPA Knowledge Sharing Effort, which is aimed at developing techniques and methodologies for building large–scale knowledge bases which are sharable and reusable. KQML is both a message format and a message–handling protocol to support run–time knowledge sharing among agents. KQML is described and evaluated as an agent communication language relative to the desiderata.

Introduction

The computational environment which is emerging in such programs as the National Information Infrastructure (NII) is characterized by being highly distributed, heterogeneous, extremely dynamic, and comprising a large number of autonomous nodes. An information system operating in such an environment must handle several emerging problems:

- The predominant architecture on the Internet, the client-server model, is too restrictive. It is difficult for current Internet information services to take the initiative in bringing new, critical material to a user's attention. Some nodes will want to act as both clients and servers, depending upon with whom they are interacting.
- Several forms of heterogeneity need to be handled, *e.g.* different platforms, different data formats, the capabilities of different information services, and the different standards (CORBA, OLE, LINDA, ISIS, ZIRCON, OpenDoc, *etc.*) used by those services.
- Many software technologies such as event simulation, applied natural language processing, knowledge–based reasoning, advanced information retrieval, speech processing, *etc.* have matured to the point of being ready to participate in and contribute to an NII environment. However, there is a lack of tools and techniques for constructing intelligent clients and servers or for building agent–based software in general.

A community of *intelligent agents* can address the first two of the problems mentioned above. When we describe agents as intelligent, we refer to their ability to: communicate with each other using an expressive communication language; work together cooperatively to accomplish complex goals; act on their own initiative; and use local information and knowledge to manage local resources and handle requests from peer agents.

Desiderata

In this section we identify 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.

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 communi-

*This work was supported in part by the Air Force Office of Scientific Research under contract F49620–92–J–0174, and the Advanced Research Projects Agency monitored under USAF contracts F30602–93–C–0177 and F30602–93–C–0028 by Rome Laboratory.

cation 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

Semantics is an issue that has often been neglected during the design of communication languages. Such neglect is the direct result of the obscurity that surrounds the purpose and the desired features of communication languages. Although the semantic description of communication languages and their primitives is often limited to natural language descriptions, a well-defined semantic description is anything but a luxury. This is especially true 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 for speed, and for 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 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.

The Knowledge Sharing Effort

The ARPA Knowledge Sharing Effort (KSE) is a consortium to develop conventions facilitating sharing and reuse of knowledge bases and knowledge–based systems. Its goal is to define, develop, and test infrastructure and supporting technology, to enable participants to build larger and more broadly functional systems than could be achieved working alone. The

KSE is organized around four working groups, each of which addresses a complementary problem identified in current knowledge representation technology: *Interlingua*, *Knowledge Representation System Specification*, *Shared, Reusable Knowledge 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 Formalism* or *KIF* (Genesereth & Fikes 1992), which is based on first order logic with extensions to support non-monotonic reason and definitions. KIF provides both a specification for the syntax of a language, and a specification for its semantics. KIF can be used to support translation from one content language to another, or as a common content language between two agents that use different native representation languages. Information about KIF and associated tools is available from `http://www.cs.umbc.edu/kse/kif/`.

The *Knowledge Representation System Specification Group* (KRSS) focuses on defining common constructs within families of representation languages. It has recently finished a common specification for terminological representations in the KL-ONE family. This document and other information on the KRSS group is available as `http://www.cs.umbc.edu/kse/krss/`.

The *Shared, Reusable Knowledge Bases Group* (SRKB) 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 as `http://www.cs.umbc.edu/kse/srkb/`.

The scope of the *External Interfaces Group* is the run-time interaction between knowledge–based systems and other modules. 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 (Pastor, McKay, & Finin 1992). The KQML language is one of the main results to come out of the external interfaces group of the KSE. General information is available from `http://www.cs.umbc.edu/kqml`.

The KQML Language

Knowledge Query and Manipulation Language (KQML) is a language that is designed to support interaction among intelligent software agents. It was developed by the ARPA–supported Knowledge Sharing Effort (Neches *et al.* 1991; Patil *et al.* 1992) and independently implemented by several research groups. It has been successfully used to implement a variety of information systems using different software architectures.

Communication takes place on several levels. The content of the message is only a part of the communication. Being able to locate and engage the attention of another agent with which an agent wishes to communicate is a part of the process. Packaging a message in a way that makes clear the purpose of an agent's communication is another.

When using KQML, a software agent transmits content messages, composed in a language of its own choice, wrapped inside of a KQML message. The content message can be expressed in any representation language and written in either ASCII strings or one of many binary notations (*e.g.* network independent XDR representations). KQML implementations ignore the content portion of a message except to recognize where it begins and ends.

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 proven to be quite flexible.

KQML is expected to be supported by a software substrate that makes it possible for agents to locate one another in a distributed environment. Most current implementations come with custom environments of this type; these are commonly based on helper programs called *routers* or *facilitators*. These environments are not a part of the KQML specification. They are not standardized and most of the current KQML environments will evolve to use one or more of the emerging commercial frameworks, such as OMG's CORBA or Microsoft's OLE2.

The KQML language simplifies its implementation by allowing KQML messages to carry any useful information, such as the names and addresses of the sending and receiving agents, a unique message identifier, and notations by any intervening agents. There are also optional features of the KQML language which contain descriptions of the content: its language, the ontology it assumes, and some type of more general description, such as a descriptor naming a topic within the ontology. These optional features make it possible for supporting environments to analyze, route and deliver messages based on their content, *even though the content itself is inaccessible.*

The forms of these parts of the KQML message may vary, depending on the transport mechanism used to carry the KQML messages. In implementations which use TCP streams as the transport mechanism, they appear as fields in the body of the message. In an earlier version of KQML, these fields were kept in *reserved* locations, in an outer wrapper of the message, to emphasize their difference from other fields. In other transport mechanisms the syntax and content of these message may differ. For example, in the E-mail implementation of KQML, these fields are embedded in KQML mail headers.

The set of performatives forms the core of the language. It determines the kinds of interactions one can have with a KQML-speaking agent. The primary function of the performatives is to identify the protocol to be used to deliver the message and to supply a *speech act* that the sender attaches to the content. The performative signifies that the content is an *assertion*, a *query*, a *command*, or any other mutually agreed upon speech act. It also describes how the sender would like any reply to be delivered (i.e., what protocol will be followed).

Conceptually, a KQML message consists of a performative, its associated arguments which include the real content of the message, and a set of optional *transport* arguments, which describe the content and perhaps the sender and receiver. For example, a message representing a query about the price of a share of IBM stock might be encoded as:

```
(ask-one
  :content (PRICE IBM ?price)
  :receiver stock-server
  :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*. A similar 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 a query is to be issued that is likely to have large number of replies, the replies can be requested one at a time using the *stream-all* performative. (To save space, we will no longer repeat fields which are the same as in the above examples.)

```
(stream-all
  ;;?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, the *standby* performative can be wrapped around the preceding message:

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

The *standby* performative expects a KQML language content; it requests that the agent receiving the request hold the stream of messages and release them one at a time. The sending agent requests a reply 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). This combination is so useful that it can be abbreviated:

```
(generate
    :content (PRICE ?VL ?price)))
```

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). An abbreviation for subscribe/stream combination is known a *monitor*.

```
(monitor
    :content (PRICE IBM ?price)))
```

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 Figure 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 asynchronous messages it is willing to handle) and *recruit* (which can be used to find suitable agents for particular types of messages). For example, the server in the above example might have earlier announced:

```
(advertise
    :ontology NYSE-TICKS
    :language LPROLOG
    :content (monitor
                :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 *monitor* message.

Basic query performatives:

- evaluate, ask-if, ask-in, ask-one, ask-all, ...

Multi-response query performatives:

- stream-in, stream-all, ...

Response performatives:

- reply, sorry, ...

Generic informational performatives:

- tell, achieve, cancel, untell, unachieve, ...

Generator performatives:

- standby, ready, next, rest, discard, generator, ...

Capability-definition performatives:

- advertise, subscribe, monitor, import, export, ...

Networking performatives:

- register, unregister, forward, broadcast, route, ...

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

How KQML Stacks Up

In this section, we evaluate the KQML language as it stands today, relative to our desiderata for agent communication languages.

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. The syntax is simple and allows for the addition of new parameters, if deemed necessary, in a future revision 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.

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 (Labrou & Finin 1994).

Implementation

The two implementations of KQML currently available, the Lockheed KQML API and the UNISYS KQML API, each provides a content–independent message router and a facilitator. Facilitators are specialized KQML agents that maintain information about other agents in their domain and those agents' query–answering capabilities (existing versions of facilitators supply only simple registration services). 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. 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. 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 (*error* and *sorry*) that can be used as responses to messages that an application cannot process or comprehend.

Conclusion

A good agent communication language has many needs, some of which are in competition. KQML is a new communication language that addresses many (although not all) of these needs. Additional information on KQML, including papers, language specifications, access to APIs, information on email discussion lists, *etc.*, can be obtained via the world wide web as `http://www.cs.umbc.edu/kqml/` and via ftp from `ftp.cs.umbc.edu` in `pub/kqml/`.

References

ARPA Knowledge Sharing Initiative. 1992. Specification of the KQML agent–communication language. ARPA Knowledge Sharing Initiative, External Interfaces Working Group working paper. Available as `http://www.cs.umbc.edu/kqml/papers/kqml-spec.ps`.

Finin, T.; McKay, D.; Fritzson, R.; and McEntire, R. 1993. KQML: an information and knowledge exchange protocol. In *International Conference on Building and Sharing of Very Large–Scale Knowledge Bases.* A version of this paper will appear in Kazuhiro Fuchi and Toshio Yokoi (Ed.), "Knowledge Building and Knowledge Sharing", Ohmsha and IOS Press, 1994. Available as `http://www.cs.umbc.edu/kqml/papers/kbks.ps`.

Finin, T.; McKay, D.; Fritzson, R.; and McEntire, R. 1994. The KQML information and knowledge exchange protocol. In *Third International Conference on Information and Knowledge Management.*

Finin, T.; Nicholas, C.; and Yesha, Y., eds. 1993. *Information and Knowledge Management, Expanding the Definition of Database.* Lecture Notes in Computer Science 752. Springer–Verlag. (ISBN 3–540–57419–0).

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

Genesereth, M. 1993. An agent–based approach to software interoperability. Technical Report Logic–91–6, Logic Group, CSD, Stanford University.

Kuokka, D. R.; McGuire, J. G.; Weber, J. C.; Tenenbaum, J. M.; Gruber, T. R.; and Olsen, G. R. 1993. Shade: Technology for knowledge–based collaborative. In *AAAI Workshop on AI in Collaborative Design.*

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

McGuire, J. G.; Kuokka, D. R.; Weber, J. C.; Tenenbaum, J. M.; Gruber, T. R.; and Olsen, G. R. 1993. Shade: Technology for knowledge–based collaborative engineering. *Journal of Concurrent Engineering: Applications and Research (CERA)* 1(2).

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

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

Pan, J. Y.-C., and Tenenbaum, J. M. 1991. An intelligent agent framework for enterprise integration. *IEEE Transactions on Systems, Man and Cybernetics* 21(6). (Special Issue on Distributed AI).

Pastor, J.; McKay, D.; and Finin, T. 1992. View–concepts: Knowledge–based access to databases. In *First International Conference on Information and Knowledge Management.*

Patil, R.; Fikes, R.; Patel-Schneider, P.; McKay, D.; Finin, T.; Gruber, T.; and Neches, R. 1992. The DARPA knowledge sharing effort: Progress report. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference.* Available as `http://www.cs.umbc.edu/kqml/-papers/kr92.ps`.

Wiederhold, G.; Wegner, P.; and Ceri, S. 1992. Toward megaprogramming. *Communications of the ACM* 33(11):89–99.