KQML & FLBC: Contrasting Agent Communication Languages

Scott A. Moore
University of Michigan Business School
samoore@umich.edu

Abstract

Communication languages for agents and otherwise have been designed to minimize the size of the message and to function more as a data-passing protocol. Little emphasis has been placed on the flexibility of the language or the transparency of the message's meaning. The author analyzes the recently defined formal semantics of KQML, which is used as the exemplar of agent communication languages. Based on this, he then specifies an FLBC message whose effects would be more or less equivalent. The purpose of this is to compare standard agent based languages (KQML in this case) with one that more directly represents the meaning of the message. The results indicate that the latter type of language makes message composition more powerful, message decomposition feasible, and has the by-product of instantly defining many more possibly useful messages.

1. Introduction

Agent communication languages (e.g., KQML (Finin et al. [1], Labrou [2]), Agent-0 (Shoham [3]), ACL (FIPA [4])), electronic data interchange (e.g., EDIFACT [5]), protocols such as Contract Net (Smith [6]), inter-application communication languages (e.g., Apple Events [7])—all provide different means for programs to send information that can be processed and responded to by other programs. Each language for automated electronic communication implements a relatively simple data passing protocol. Further, in most cases, the message's meaning is only minimally restricted by the message's surface structure and is obviously determined by convention. An extreme example of this type of messaging is the following: I have a pager. When I receive a message that consists of only a "1", this tells me that I am supposed to call my wife at home when it is convenient but that there is no rush. Clearly, the digit "1" does not contain this information—the meaning is determined by convention. And so it is for these languages for automated electronic communication.

What I propose here is that a message for automated electronic communication should more closely reflect its underlying meaning. The meaning must still be determined conventionally but that meaning should be more transparent and more amenable to computation, leading to development of communication-based applications that is faster and more wide-spread than current practice. I take KQML, a well-known agent communication language, as an exemplar of the current practice and FLBC, a general purpose automated electronic communication language, as an exemplar of the proposed type of language.

Since researchers have only begun to investigate this latter approach, its benefits when viewed in the context of relatively complex communication needs are not clear. This paper takes a detailed look at how this approach works when confronted with the more extensive needs of a system such as KQML. Its standard messages, having evolved over several years among many researchers in several projects, can be seen as describing at least a significant portion of what is currently believed to be the communication needs of agents. Recently Labrou defined the formal semantics for KQML (Labrou [2]; a subset of this is presented by Labrou & Finin [8]). This clarifies many problems of the prior definition and provides a good test bed for the current investigation. I translate all thirty KQML messages—what are referred to in the literature as KQML performatives—into more or less equivalent FLBC representations. The results are that KQML can be rather directly translated into FLBC. Translating these messages highlights each language's operating assumptions and how they significantly affect how appropriate each is for electronic commerce.

2. Languages for automated electronic communication

In this section I provide a brief overview of KQML and FLBC, highlighting those aspects that are most applicable to the translations and analysis that follow.

2.1. KQML

KQML, the Knowledge Query & Manipulation Language, has been developed under a DARPA funded project and is probably the most well-known and widely-implemented agent communication language. (More information and references can be found at their web site, `http://www.cs-.umbc.edu/kqml/`.) It has a LISP-style syntax and has

some basis in speech act theory. A KQML message generally has the form

```
(perfName
  :sender A     :receiver B
  :content X    :language L   :ontology N
  :reply-with W :in-reply-to P)
```

This is a message from A to B in reply to a previous message identified by P. Any message sent in response to this message should include `:in-reply-to W`. The content X has a syntax like that specified by the language L (e.g., Prolog or KIF) whose terms are taken from ontology N. The message's meaning is determined by the combination of the performative `perfName` and the content X. The performative has values such as `ask-if`, `tell`, and `insert`. The content of these messages detail what is asked, told, or inserted.

The semantics of the KQML primitives are defined only within a highly restricted communication environment. Most exchanges must be preceded by an `advertise` performative which states that the sender is interested in receiving messages of a certain kind [2, p. 114]. The `advertise` performative establishes the necessary precondition in the message's receiver—that is, a belief that the sender of the `advertise` wants to receive a certain type of message. This precondition enables the receiver of the `advertise` to send, at some appropriate time in the future, a message back to the original sender.

For example, in order for agent B to process the `ask-all(A, B, X)` performative—a message from A to B asking about X—, agent B must have an intent to process this performative with this content from agent A and agent A must know of this intent [2, p. 90]. A preceding `advertise(B, A, process(ask-all(A, B, X)))` performative would establish—in fact, is the *only* way to establish—knowledge in agent A of agent B's intent and obligation to process this performative. Since preconditions for sending performatives *must* be established by a performative [2, p. 111] and since agents are assumed to be truthful [2, p. 84], an agent can *only* send performatives to another agent if that agent asked for it (with a few special exceptions, see [2, p. 114]). A published portion of Labrou's dissertation glosses over this major restriction to KQML's functionality (see [8, p. 451, footnote 15]).

Labrou asserts that "[i]t is always an implicit postcondition that `know(A, process(B, M))`, for any M that A sent to B, unless a `sorry` or `error` is sent in response to M. In other words, a performative is always delivered to its destination ... and the sending agent knows that." [2, p. 93] Implicit in this statement is that delivering a performative to its destination and the processing of that performative are the same thing. This equivalence results from the practices of 1) agents advertising the performatives that they will process, and 2) other agents sending performatives only to agents that have advertised that they will process those performatives. The question remains: What does it mean for a KQML agent to process something as opposed to simply receiving it? This is unclear from the literature.

The minimal emphasis given to it in that paper leads to the conclusion that KQML's developers do not see this as a serious problem. Again: agent A cannot ask agent B a question unless agent B has given permission to agent A that it can ask that question. KQML's developers might see this as a *feature* of the language rather than a restriction. It makes processing incoming messages much simpler than it would otherwise be. For example, consider an incoming `insert(A, B, X)` performative. The effect on B is that it believes that X. This is amazing! Just by sending a message, A can directly affect B's beliefs. However, the amazement disappears when it is recalled that B previously asked (i.e., `advertised`) A to send this information. It certainly seems reasonable that B should believe what A tells it to believe since B asked for it in the first place.

2.2. FLBC

FLBC (Kimbrough & Moore [9], Moore [10], Moore & Kimbrough [11]), the Formal Language for Business Communication, is a language for automated electronic communication based on speech act theory (Austin [12], Bach & Harnish [13]) that allows more complex message structures than most languages. (More information can be found at the web site `http://www-personal-.umich.edu/~samoore/research/flbc/`.) An FLBC message is an XML document (Cover [14], Light [15]) and has deep foundations in speech act theory. (For more details on this, see Moore [16] and Moore & Kimbrough [9]. Also, the `flbcMsg` DTD is located at `http://www-personal.umich.edu-/~samoore/research/flbcMsg.dtd`.) An FLBC message has the form

```
<flbcMsg msgID="M">
  <simpleAct speaker="A"  hearer="B">
    <illocAct force="F"  vocab="N"
              language="L">
      X
    </illocAct></simpleAct>
  <context  respondingTo="V"></context>
</flbcMsg>
```

The interpretation of this message is fairly straightforward. This is a message, identified by M, from A to B in reply to a previous message identified by V. Speech act theory hypothesizes that all utterances have the form `F(P)` where F is the *illocutionary force* of the message (e.g., `inform`, `request`, `predict`) and P is the propositional content of the message (what is being informed, requested, or predicted). In conformance with the theory's `F(P)` hypothesis, the structure of this and all FLBC messages is `F(X)`. The

content X is written using the language L and the ontology N.

The real difference between KQML and FLBC lies in the distinction between performatives and illocutionary forces. With performatives, saying it makes it so—think "I christen thee..." or "You're out!" That's how KQML works as well. Sending an `insert` to another agent results in that agent always inserting data into its knowledge base—if an associated `advertise` was previously sent—unless that agent simply could not understand the data. The FLBC is designed to operate in a much less restrictive environment, albeit one that requires much more processing of incoming messages and a more complex knowledge base. The standard effects of an FLBC message define the minimal possible effects it will have on the receiver of the message. (See Moore [16] for a complete discussion of standard effects.) For example, the effects of A informing B that X are that

```
considerForKB A believes X,
considerForKB A believes (B not
  believes X),
considerForKB A wants (B believes X)
```

If these three are believed by B, then it will believe that 1) A believes X, 2) A believes that B does not believe that X (which is not the same as B believing that not X), and 3) agent A wants B to believe that X. However, all three of these effects on B must be considered, and possibly rejected, by B before they are actually added to its knowledge base. Agent B may not believe A because it does not know A or does not trust A or has evidence that A believes not X. With an FLBC system, B is merely expected to *consider* for its knowledge base that A wants B to believe that A believes X (and so on). This leaves open the possibility that B eventually ignores or otherwise discounts A's assertion.

3. Translating the performatives

In this section I translate several of KQML's performatives into more or less equivalent FLBC messages. (The rest of the performatives and their translations can be found in the full version of this paper which can be downloaded from `http://www-personal.umich.edu-/~samoore/research/flbc/`.) At the beginning of each section I list the pages of the formal definition which that section draws from. For each KQML performative I provide, directly from the language definition [2], the message format, the informal description of what the message means, the formal description of the same, the preconditions if provided for successfully sending (for the sender) and processing (for the receiver) this message, and the postconditions if provided of a successful sending and a successful processing. The last three are expressed in terms of the operators listed in Appendix C. After this KQML-related information, I display the FLBC message that has the same effects, as defined by FLBC's standard effects (Moore [16]), as the KQML performative. Even though KQML's formal definition is essentially unpublished, the reader can verify these translations since this paper contains all, or at least a significant portion of, the information needed to determine the meaning of each KQML performative. I also provide an informal translation of the FLBC message, the standard effects of the FLBC message, and an informal translation of these standard effects. I also list the vocabulary that must be added to FLBC's standard vocabulary in order to send messages that have the meaning of KQML's performatives. These additions are collected in Appendix B. I conclude each section with a discussion of that performative and its translation.

In these translations for the purposes of compactness I leave out from the KQML representation the parameters `:sender A`, `:receiver B`, `:language L`, and `:ontology N`. These are directly and non-controversially represented in the FLBC message. Where it does not confuse the issue I also leave out `:in-reply-to` and `:reply-with`. For FLBC messages I leave out the opening and closing `<flbcMsg>` and `</flbcMsg>` tags. I also do not translate the content of the KQML message into an FLBC representation since it is represented in a formalism separate from the KQML language (e.g., KIF). That is, the KQML language does not contain terms to represent the content of its messages so there is nothing to translate.

3.1. Discourse performatives

These performatives are used for information and knowledge exchange.

`ask-if` [2, p. 39, 89]

KQML `(ask-if :content X)`

Description "A wants to know what B believes regarding the truth status of the content X."

Formal description "`want(A, know(A, Y))`, where Y may be any of `bel(B, X)`, `bel(B, not X)` or `not bel(B, X)`"

Preconditions

A "`want(A, know(A, Y))` $\wedge$ `know(A, intend(B, process(B, ask-if(A, B, X)`"

B "`intend(B, process(B, ask-if(A, B, X)))`"

Postconditions

A "`intend(A, know(A, Y))`"

B "`know(B, want(A, know(A, Y)))`"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="question">
  isTrueFalse(X)
 </illocAct></simpleAct>
```

Informal FLBC A asks B if it is true that X.

Standard effects

```
considerForKB A wants do(B,
    determine(isTrueFalse(X), T))
considerForKB A wants do(B,
    flbcMsg(B, A, inform, T))
```

Translation of standard effects on B A wants B to believe that A wants B to determine if X is true, and then for B to inform the speaker of this truth value.

Required vocabulary None.

The `ask-if` KQML performative has a simple translation of, essentially, tell me if you think this is true. The preconditions require that, yes, A must want to know about X but it also requires that A know that B intends to process A's `ask-if` message. Not only must A know about this intention but B must also actually have this intention if B is to end up processing the message. (Appendix A contains an explanation of an ambiguity in this message's definition.)

The FLBC message is a `question` from A to B. This is essentially the same as the description given above for the KQML `ask-if` performative. The KQML performative results in the recipient knowing that the sender wants to know something. The FLBC message possibly results in the recipient knowing that the sender wants the recipient to determine if something is true or false and then for the recipient to inform the sender of the truth of the statement. The FLBC specification is more detailed than KQML's. A difference is that the effects of the FLBC message are the standard effects of the question illocutionary force. As discussed above in §2.2, standard effects are possible effects that must be considered by agent B before it adds it to its knowledge base.

`ask-all` [2, p. 39, 89–90]

KQML
```
(ask-all
   :content <expression>)
```

Description "A wants to know all of B's responses that make X true of B."

Formal description "`want(A, know(A, Z))`, where Z may be 1) `bel(B, Y)`, where Y is the finite collection of all possible $Y_1, Y_2, \ldots$, where each Y_i is an instance of X with values for the variables in X and each Y_i appears once in this collection (the collection might be empty), or 2) not `bel(B, X))`, or, finally, 3) `bel(B, not X)`."

Preconditions

A "`want(A, know(A, Z))` $\wedge$ `know(A, intend(B, process(B, ask-all(A, B, X))))`"

B "`intend(B, process(B, ask-all(A, B, X)))`"

Postconditions

A "`intend(A, know(A, Z))`"

B "`know(B, want(A, know(A, Z)))`"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="request">
  <andMsg>
   <predSt>
    do(B, evaluate(bel(B,
     makeValue(J, all,
      truthStatus(X, true)))))
   </predSt>
   <predSt>
    do(B, sendMsg(B, A, inform,
     bel(B, makeValue(J,
      all(allAtOnce),
      truthStatus(X, true))),
     [respondingTo(RW)]))
   </predSt>
  </andMsg>
 </illocAct></simpleAct>
```

Informal FLBC The FLBC message is a two-part request. It is a request by A to B that 1) B determine (`evaluate`) all the values (`J`) that B believes make X true, and 2) B inform A of all the values (all in one message) that B believes make X true. This is reflected in the FLBC translation. (The vocabulary translations are in Appendix B.) This is a `request` of a complex content whose form is `<andMsg><predSt>X-</predSt><predSt>Y</predSt></andMsg>`. The first predicate (i.e., the first `<predSt>` term within the `<andMsg>` term) is the `do()` term which is translated as "B evaluate that B believes that the values in J provides `all` the values that make X `true`." Less stiltedly and as a request this is translated as "A requests that B determine all the values that it believes make X true." The second predicate is, again, a `do()` term which is translated as "B sends a message to A informing A, all in one message, of all the values [all of the Xs] that make X `true`."

Standard effects

```
considerForKB A wants do(B,
```

```
  evaluate(bel(B, makeValue(J, all,
    truthStatus(X, true)))))
considerForKB A wants (B wants do(B,
  evaluate(bel(B, makeValue(J, all,
    truthStatus(X, true)))))))
considerForKB A wants do(B,
  sendMsg(B, A, inform, bel(B,
    makeValue(J, all(allAtOnce),
      truthStatus(X, true)))))
considerForKB A wants (B wants do(B,
  sendMsg(B, A, inform, bel(B,
    makeValue(J, all(allAtOnce),
      truthStatus(X, true)))))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to determine all the values that make X true; 2) A wants B to want to determine all the values that make X true; 3) A wants B to inform A all at once of all the values that make X true; 4) A wants B to want to inform A all at once of all the values that make X true.

Required vocabulary `do/2`, `evaluate/1`, `bel/2`, `makeValue/3` (J must have either be a set of values or have the value `noBelief` or `false`), `sendMsg/5`, `truthStatus/2`

The `ask-all` performative is a request to find out *all* of the values (the Z's) that both have the same form as a certain term (the X) and are true. The message's effect on the recipient are similar to those of an `ask-if` performative: the recipient knows that the sender wants to know the set of values that makes something true. The difference between `ask-if` and `ask-all` is that for `ask-if` the item asked about is ground while for `ask-all` the item has some free variables.

ask-one [2, p. 40, 90]

KQML
```
(ask-one
  :content <expression>)
```

Description "Everything said about `ask-all` holds for `ask-one` also, but in this case A is interested in receiving exactly *one* response althought there might be more than one response to its query."

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="request">
  <andMsg>
   <predSt>
    do(B, evaluate(bel(B,
     makeValue(J, one,
      truthStatus(X, true))))
   </predSt>
   <predSt>
    do(B, sendMsg(B, A, inform,
      bel(B, makeValue(J, one,
       truthStatus(X, true))),
      [respondingTo(RW)]))
   </predSt>
  </andMsg>
 </illocAct></simpleAct>
```

Required vocabulary `do/2`, `evaluate/1`, `bel/2`, `makeValue/3` (J must have either be a set of values or have the value `noBelief` or `false`), `sendMsg/5`, `truthStatus/2`

Informal FLBC A asks B to determine a value that B believes makes X true, and to inform B what that value is.

Standard effects

```
considerForKB A wants do(B,
  evaluate(bel(B, makeValue(J, one,
    truthStatus(X, true)))))
considerForKB A wants (B wants do(B,
  evaluate(bel(B, makeValue(J, one,
    truthStatus(X, true)))))))
considerForKB A wants do(B,
  sendMsg(B, A, inform, bel(B,
    makeValue(J, one,
      truthStatus(X, true)))))
considerForKB A wants (B wants do(B,
  sendMsg(B, A, inform, bel(B,
    makeValue(J, one,
      truthStatus(X, true)))))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to determine a value that makes X true; 2) A wants B to want to determine a value that makes X true; 3) A wants B to inform A of a value that makes X true; 4) A wants B to want to inform A of a value that makes X true.

Required vocabulary `do/2`, `evaluate/1`, `bel/2`, `makeValue/3`, `truthStatus/2`, `sendMsg/5`

The above "Description" is all that is provided by Labrou. The FLBC message differs from the one given for `ask-all` only in that the symbol `all` and the predicate `all/2` are replaced by the symbol `one`.

stream-all [2, p. 45, 90]

KQML
```
(stream-all
  :content <expression>)
```

Description "Everything mentioned for `ask-all` holds for `stream-all`, too. The only difference is in the expected delivery format of the response. ... [T]he elements of the would-be collection are to be delivered one by one (using `tell` since they are statements of fact for B)."

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="request">
  <andMsg>
   <predSt>
    do(B, evaluate(bel(B,
     makeValue(J, all,
      truthStatus(X, true)))))
   </predSt>
   <predSt>
    do(B, sendMsg(B, A, inform,
     bel(B, makeValue(J,
      all(oneAtATime),
      truthStatus(X, true))),
     [respondingTo(RW)]))
   </predSt>
  </andMsg>
 </illocAct></simpleAct>
```

Informal FLBC A asks B to determine all the values that B believes make X true, and to inform B what those values are one at a time.

Standard effects

```
considerForKB A wants do(B,
  evaluate(bel(B, makeValue(J, all,
    truthStatus(X, true)))))
considerForKB A wants (B wants do(B,
  evaluate(bel(B, makeValue(J, all,
    truthStatus(X, true)))))))
considerForKB A wants do(B,
  sendMsg(B, A, inform, bel(B,
    makeValue(J, all(oneAtATime),
      truthStatus(X, true)))))
considerForKB A wants (B wants do(B,
  sendMsg(B, A, inform, bel(B,
    makeValue(J, all(oneAtATime),
      truthStatus(X, true)))))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to determine all the values that make X true; 2) A wants B to want to determine all the values that make X true; 3) A wants B to inform A one at a time of all the values that make X true; 4) A wants B to want to inform A one at a time of all the values that make X true.

Required vocabulary `do/2`, `evaluate/1`, `bel/2`, `makeValue/3` (`J` must either be a set of values or have the value `noBelief` or `false`), `sendMsg/5`, `truthStatus/2`

The above "Description" is all that is provided by Labrou. The FLBC message differs from the one given for `ask-all` only in that the symbol `allAtOnce` is replaced by the symbol `oneAtATime`. The first `do/2` terms in the FLBC translations of both `ask-all` and `stream-all` are exactly the same. They should be since the two performatives are supposed to calculate the same answer—the only difference is how the answer is delivered. This similarity is not apparent in the KQML performatives themselves.

`eos` [2, p. 45, 90–91]

KQML `(eos :in-reply-to <word>)`

Description "[I]ts only purpose is to notify B that there are no more ... *positive* response[s] to a `stream-all`. This performative is just an end-of-stream marker."

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="inform">
  <predSt>
   complete(do(A, respondingTo(RW)))
  </predSt>
 </illocAct></simpleAct>
<context respondingTo="RW" />
```

Informal FLBC A informs B that A is done responding to the message identified by RW.

Standard effects

```
considerForKB A believes
  complete(do(A, respondingTo(RW)))
considerForKB A believes (B not believes
  complete(do(A, respondingTo(RW))))
considerForKB A wants (B believes
  complete(do(A, respondingTo(RW))))
```

Translation of standard effects on B A wants B to believe that 1) A believes that A is done responding to RW; 2) A believes that B does not believe that A is done responding to RW; 3) A wants B to believe that A is done responding to RW.

Required vocabulary `complete/1`, `do/2`

In contrast with the above definitions of `ask-one` and `stream-all` whose formal definitions Labrou does not give because they are derivative, Labrou gives no formal definition of `eos` at all. The FLBC translation informs A that B is done responding to the previous message RW.

4. Results & analysis

The previous section contains five of the thirty predefined KQML performatives and their approximate FLBC translations. The results for translating all the messages

are summarized in Table 1. The performatives are listed in twenty-five columns, with five columns representing related pairs of performatives (e.g., `ask-all` and `ask-one`). The first ten rows contain the new FLBC terms needed to represent the KQML performatives. The next five rows are also needed for this purpose but their names (and approximate definitions) are shared with KQML performatives. They are separated because even though the FLBC system must define these terms, the KQML system has to provide definitions for them as well. The last ten rows list the ten (out of the approximately twenty-five previously defined) FLBC illocutionary forces that are needed to represent the KQML performatives.

This table shows that fifteen predicates needed to be created so that FLBC could represent the content of the thirty pre-defined KQML performatives. The ten illocutionary forces did not have to be created because they are already defined for FLBC.

Notice the similar structure and nearly identical content of the FLBC messages for the performatives `ask-all`, `ask-one`, and `stream-all`. This reflects the underlying similarity of the meaning of the performatives. With the KQML messages it is impossible to determine the similarity of the messages. Their meaning is determined strictly by convention so their meanings cannot be discerned from their structure.

As might have been expected, many (eighteen of thirty) of the KQML performatives are either (or both) an `inform` or a `request` (and no other force). This is consistent with Moore's findings in other automated communication languages [10]. Other findings that are consistent with findings from that paper: 1) Many illocutionary forces are not represented. 2) The `F(P)` structure seems useful and appropriate for representing the messages. 3) No additional illocutionary forces needed to be defined. 4) KQML is another standard that defines messages that include composed illocutionary forces. This ability to compose forces is unrestricted in FLBC while KQML's rules of formation place strict limits. `Advertise`, `subscribe`, `standby`, `forward`, `broadcast` and the facilitation performatives are the only performatives that may contain a `<performative>` in its content. [2, p. 43]

The main benefit of the FLBC representation scheme is that new messages can be easily defined by combining existing forces and predicates. Figure 1 contains several examples using the forces and predicates needed for the KQML performatives.

The communication style underlying these two languages makes each appropriate for different types of applications. KQML is more appropriate for trusting, relatively small society of agents while FLBC seems better designed for larger, less trusting and cooperative set of agents. KQML nearly requires that the group of agents trust each other highly. Consider the postconditions of `B` for `insert`: `bel(B, X)`. Merely by sending an `insert` to `B`, `A` has convinced `B` that

1. Report that an agent was subscribed (`report(B, A, do(B, subscribe(A, M)))`)
2. Deny that an agent did something (`denial(B, A, do(B, subscribe(A, M)))`)
3. Promise that an agent will do something (`promise(B, A, do(B, subscribe(A, M)))`)
4. Promise that an agent will complete something (`promise(B, A, complete(do(B, subscribe(A, M))))`)
5. Predict that an agent will complete something (`predict(B, A, complete(do(B, subscribe(A, M))))`)
6. Request that someone ask someone else if something is true (`request(B, A, question(A, C, X))`)
7. Deny that the speaker can undo something (`denial(B, A, undo(A, X))`)
8. Question if the other agent is done responding to a request (`question(B, A, complete(do(A, respondingTo(X))))`)
9. Promise that an agent will undo something (`promise(B, A, undo(A, X))`)

Figure 1: New messages implemented out of predicates defined for KQML and existing illocutionary forces

`B` believes that `X`. According to the preconditions for `B`, `B` must want to process the `insert` from `A`. This precondition will only arise by `B` telling `A` that `B` will process the `insert` message. This style of requiring a preceding message promising that a message will be processed permeates the KQML language. (See translations and discussions in the previous section.)

FLBC's structure—cleanly separating the illocutionary force from its content—also encourages the construction of messages whose meaning is more directly represented by and apparent in the message's content. This paper is part of the evidence supporting this assertion.

Other information File: HICSS99SAM.tex, 9/30/1998.

A. Problem with the KQML formal definitions

There is a difficulty in the formal description, preconditions, and postconditions for the `ask-if` performative. The difficulty arises from the `Y` variable. This variable can have exactly one of three different values. However, the `want(A, know(A, Y))` predicate is supposed to represent a particular *mental state* (his term) for the `A` agent. The mental state cannot have an uninstantiated variable (`Y`) in it—the agent cannot just *want*, it must want something. The mental state

Table 1: Need for illocutionary forces and new predicates for each KQML performative

Term	ask-if	ask-one, -all	stream-all	eos	tell	untell	deny	insert	uninsert	delete-one, -all	undelete	achieve	unachieve	advertise	subscribe	error	sorry	register	unregister	transport-address	forward	broadcast	broker-one, -all	recommend-one, -all	recruit-one, -all
Performatives																									
New predicates																									
complete				■													■								
do		■	■	■								■			■		■				■	■	■	■	■
evaluate		■	■																		■	■		■	
isAbleTo																							■	■	
makeValue		■	■																					■	■
not						■										■									
process														■		■								■	
responseTo																							■		
truthStatus		■	■																					■	■
undo											■		■						■						
Predicates shared with KQML																									
bel		■	■		■	■															■			■	■
forward																		■	■		■				
register																		■							
sendMsg		■	■																						
subscribe															■										
Illocutionary forces																									
assert								■			■														■
denial							■																		
describe										■										■					
dispute																									
inform		■	■	■	■	■											■						■	■	■
promise														■											
question	■															■									
report											■					■									
request		■	■	■	■	■					■	■	■		■			■	■				■	■	■
retract									■										■	■					

must be the want/2 predicate with the Y variable instantiated to *some* value. A choice (that I discard) for replacing this "faulty" predicate is that of substituting any one of the three possible predicates for Y—e.g., want(A, know(A, *bel(B, not X)*)). Suppose that agent A finds out that B believes X. It does not seem obvious that this new piece of information would satisfy this particular want(). Given the awkwardness of this particular solution, others should be considered.

Given the text of the definition, the somewhat obvious choice would be to give Y the value want(A, know(A, bel(B, X) ∨ bel(B, not X) ∨ not bel(B, X))); however, Labrou states that he does not allow disjunctions within the scope of his epistemic operators [2, p. 88]. Given this, the next choice would be to have a disjunction of wants, resulting in the following definitions:

Formal description

```
want(A, know(A, bel(B, X))) ∨
want(A, know(A, bel(B, not X))) ∨
want(A, know(A, not bel(B, X)))
```

Preconditions **A** (want(A, know(A, bel(B, X))) ∨ want(A, know(A, bel(B, not X))) ∨ want(A, know(A, not bel(B, X)))) ∧ know(A, intend(B, process(B, ask-if(A, B, X))))

B intend(B, process(B, ask-if(A, B, X))). This is the same as given above.

Postconditions **A** intend(A, know(A, bel(B, X))) ∨ intend(A, know(A, bel(B, not X))) ∨ intend(A, know(A, not bel(B, X)))

B know(B, want(A, bel(B, X))) ∨ know(B, want(A, bel(B, not X))) ∨ know(B, want(A, not bel(B, X)))

Though this is more difficult to understand than what Labrou presented, it unambiguously describes the mental states of the agents. It is not so clear if this is a helpful or easily implementable formalization of or even an accurate description of the reality of the situation. For example, consider the preconditions for agent A. This states that the agent has one (or more) of three separate desires (e.g., wants). This does not seem right. More probably, and more similar to the *form* of the predicate proposed by Labrou, agent A has one desire—to "know what B believes regarding the truth status of the content X." [2, p. 89] A new predicate can be defined to reflect what the *informal* description says, and not what Labrou's *formal* description says that it says:

want(A, know(A, bel(B, valOf(ts, truthStatus(X, ts)))))

This introduces the predicates valOf/2 and truthStatus/2. The above term can be translated as "A wants to know what B believes the truth status of X is" or, more directly, "A wants A to know what B believes the value of ts is in the statement 'the truth status of X is ts'". In the predicate and in these statements ts is simply a placeholder. The letters chosen are not significant; what is significant are that the symbol in the first argument of valOf/2 appears somewhere in the second argument.

For this newly expressed desire to be satisfied, A must come to possess knowledge of the form

know(A, bel(B, truthStatus(X, J)))

where J takes one of the values true, false, or noBelief. These three values correspond to the three possible values for Y in Labrou's definition.

Given the above redefinition, the three items should have the following definitions:

Formal description

```
want(A, know(A, bel(B, valOf(ts,
     truthStatus(X, ts)))))
```

Preconditions **A** want(A, know(A, bel(B, valOf(ts, truthStatus(X, ts))))) ∧ know(A, intend(B, process(B, ask-if(A, B, X))))

B intend(B, process(B, ask-if(A, B, X))). This is the same as given above.

Postconditions **A** intend(A, know(A, bel(B, valOf(ts, truthStatus(X, ts)))))

B know(B, bel(B, truthStatus(X, J)))

This redefinition overcomes my objections to Labrou's formulation: Agent A's want() is unambiguous and its formal representation closely reflects the informal one.

B. New FLBC predicates

The following are descriptions of each of the predicates that had to be added to the FLBC's vocabulary.

bel(A, B) A believes B.

complete(A) A is complete

do(A, B) A does task B

evaluate(A) Evaluate expression A

forward(A, B) A forwards message B

isAbleTo(A, B) A is able to B

makeValue(A, B, C) The values in A (which must either be a set of values or have the value `noBelief` or `false`) provide B solutions (which must be either `one` or `all`) to C

not(A) It is not the case that A

process(A, B) A processes B

register(A, B) A registers for B

responseTo(A) The response to A

sendMsg(A, B, C, D, E) A sends a message to B with force C, content D, and context E. This is essentially equivalent to the `sendMSG` predicate defined by Labrou [2, p. 98].

subscribe(A, B) A subscribes to B

truthStatus(A, B) The truth status of A is B (which must be either `true` or `false`

undo(A, B) A reverses the effects of B

C. KQML predicates

Labrou uses the following predicates to "describe mental states of agents that use speech acts" [2, p. 83].

bel(A, P) "P is true for A."

know(A, P) "expresses a state of knowledge awareness of A, about the state P."

want(A, P) "agent A desires the event (or state) described by P, [sic] to occur."

intend(A, P) "A has every intention of doing P and thus is tommitted to a course of action towards achieving P in the future."

References

[1] T. Finin, Y. Labrou, and J. Mayfield, "KQML as an agent communication language," in *Software Agents* (J. M. Bradshaw, ed.), ch. 14, pp. 291–316, AAAI Press/The MIT Press, 1997.

[2] Y. Labrou, *Semantics for an Agent Communication Language*. 1996, University of Maryland, Computer Science and Electrical Engineering Department, August 1996. UMI Number 9700710.

[3] Y. Shoham, "Agent-oriented programming," *Artificial Intelligence*, vol. 60, pp. 51–92, 1993.

[4] Foundation for Intelligent Physical Agents, "FIPA 97 specification part 2: Agent communication language," November 28, 1997. Geneva, Switzerland.

[5] UN/EDIFACT Rapporteurs, "United nations directories for electronic data interchange for administration, commerce and transport." `http://www.unicc.org/unece-/trade/untdid/Welcome.html`, downloaded on June 22, 1998.

[6] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on Computers*, vol. C-29, December 1980.

[7] Apple Computer, Inc., "Apple Event registry: Standard suites." `http://dev2.info.apple.com-/FTPIndices/App-3.html`, downloaded on August 13, 1996.

[8] Y. Labrou and T. Finin, "A semantics approach for KQML — a general purpose communication language for software agents," in *Proceedings of the Third International Conference on Information and Knowledge Management*, (Gaithersburg, MD), pp. 447–55, National Institute of Standards and Technology, ACM Press, November 29–December 2, 1994 1994.

[9] S. O. Kimbrough and S. A. Moore, "On automated message processing in electronic commerce and work support systems: Speech act theory and expressive felicity," *ACM Transactions on Information Systems*, vol. 15, pp. 321–367, October 1997.

[10] S. A. Moore, "Categorizing automated messages." Forthcoming in *Decision Support Systems*, 1998.

[11] S. A. Moore and S. O. Kimbrough, "Message management systems at work: Prototypes for business communication," *Journal of Organizational Computing*, vol. 5, no. 2, pp. 83–100, 1995.

[12] J. L. Austin, *How To Do Things With Words*. Harvard University Press, 2nd ed., 1975.

[13] K. Bach and R. M. Harnish, *Linguistic Communication and Speech Acts*. MIT Press, 1979.

[14] R. Cover, "Extensible markup language (XML)." Accessed at `http://www.sil.org/sgml/xml.html`, June 22, 1998.

[15] R. Light, *Presenting XML*. SAMS.net Publishing, 1st ed., 1997.

[16] S. A. Moore, "A foundation for flexible automated electronic commerce." Working paper #98014 at the University of Michigan Business School, August 1998.