KQML & FLBC: Contrasting agent communication languages*

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

October 1, 1998

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.

*This is a draft. Do not reference. A shorter version of this paper was published in HICSS-32.

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., `advertise`d) 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 all of KQML's performatives into a more or less equivalent FLBC representation. 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))` ∧ `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, ..., 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)) ∧ 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`.

---

tell [2, p. 45+, 91]

KQML
```
(tell
        :content <expression>)
```

Description "A states to B that A believes the content to be true."

Formal description "`bel(A, X)`"

Preconditions

A "$\texttt{bel(A, X)} \land \texttt{know(A, want(B, know(B, bel(A, X))))}$"

B "`intend(B, know(B, bel(A, X)))`"

Postconditions

A "`know(A, know(B, bel(A, X)))`"

B "`know(B, bel(A, X))`"

FLBC

```
<simpleAct speaker="A" hearer="B">
  <illocAct force="inform">
    <predSt>
      bel(A, X)
    </predSt>
  </illocAct></simpleAct>
```

Informal FLBC `A` informs `B` that `A` believes it is the case that `X`.

Standard effects

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

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

Required vocabulary `bel/2`

The FLBC message consists of A informing B that A believes X. One of the standard effects of this message on B are that A wants B to believe that A believes X or, in the terminology used above, `considerForKB intend(A, know(B, bel(A, X)))`. Notice the similarity to the postcondition for B.

---

untell [2, p. 47, 91]

KQML `(untell :content <expression>)`

Description "A states to B that A believes the negation of the content to be true." [2, p. 91] This is an editing mistake. Elsewhere Labrou asserts that the `untell` performative "indicates that the `:content` expression in [sic] not true of the sender, *i.e.,* it is not part of the sender's VKB [virtual knowledge base]." [2, p. 47] This is more consistent with his presentations and examples throughout the book and is the one I use.

Formal description "¬(bel(A, X))"

Preconditions

A "¬(bel(A, X)) ∧ know(A, want(B, know(B, ¬(bel(A, X))))"

B "intend(B, know(B, ¬(bel(A, X)))"

Postconditions

A "know(A, know(B, ¬(bel(A, X))))"

B "know(B, ¬(bel(A, X)))"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="inform">
  <predSt>
   not bel(A, X)
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC A informs B that it is not the case that A believes that X.

Standard effects

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

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

Required vocabulary `bel/2`, `not/1`

The preconditions for B are that B wants to know if it is not the case that A believes some fact.

---

deny [2, p. 47–48, 91–92]

KQML `(deny :content <expression>)`

Description "A states to B that A believes the negation of the content to be true."

Formal description "bel(A, ¬(X))"

Preconditions

A "bel(A, ¬(X)) ∧ know(A, want(B, know(B, bel(A, ¬(X)))))"

B "intend(B, know(B, bel(A, ¬(X))))"

Postconditions

A "know(A, know(B, bel(A, ¬(X))))"

B "know(B, bel(A, ¬(X)))"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="denial">
  <predSt>
   X
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC `A` denies to `B` that `X`.

Standard effects

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

Translation of standard effects on `B` `A` wants `B` to believe that 1) `A` believes that not `X`; 2) `A` believes that `B` does not believe that not `X`; 3) `A` wants `B` to believe that not `X`.

Required vocabulary None.

As Labrou points out, `deny(A, B, X)` is equivalent to `tell(A, B, ¬(X))`. [2, p. 92] For FLBC, `denial(A, B, X)` is equivalent to `inform(A, B, X)`.

---

insert [2, p. 48, 92–93]

KQML
```
(insert
     :content <expression>)
```

Description "A wants B to believe the content (in other words, to make the content a part of its knowledge store)."

Formal description "`want(A, bel(B, X)))`"

Preconditions

A "`know(A, intend(B, process(B, insert(A, B, X))))`"

B "`intend(B, process(B, insert(A, B, X)))` ∧ `¬(bel(B, X))`"

Postconditions

A "`know(A, bel(B, X))`"

B "`bel(B, X)`"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="assert">
  <predSt>
   X
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC `A` asserts to `B` that `X`.

Standard effects

```
considerForKB A believes X
considerForKB A wants (B believes X)
```

Translation of standard effects on `B` `A` wants `B` to believe that 1) `A` believes that `X`; 2) `A` wants `B` to believe that `X`.

Required vocabulary None.

FLBC requires that the effects of this message be `considerForKB A believes X` and `A wants (B believes X)`; thus, the hearer must at least consider both that the speaker believes that `X` and that the speaker wants the hearer to believe that `X`. KQML is quiet on requiring that the hearer must believe that the speaker believes that `X` but it does require that the hearer believe that `X`.

---

uninsert [2, p. 48–49, 93]

KQML
```
(uninsert
     :content <expression>)
```

Description "A wants B to remove X from its knowledge store, where X was inserted to B's knowledge store upon A's request in the past."

Formal description "`want(A, ¬(bel(B, X)))`"

Preconditions

A "`know(A, process(B, M))` where `M` is `insert(A, B, X)`"

B "`process(B, M))` ∧ `bel(B, X)`"

Postconditions

A "`know(A, ¬(bel(B, X)))`"

B "`¬(bel(B, X))`"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="retract">
  <predSt>
   X
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC A retracts to B that X.

Standard effects

```
considerForKB A not believes X
considerForKB A believes (B believes
  time(before(now), A believes X))
considerForKB A wants (B not believes X)
```

Translation of standard effects on B A wants B to believe that 1) A does not believe that X; 2) A believes that B believes that A has believed that X; 3) A wants B to not believe that X.

Required vocabulary None.

The preconditions are somewhat different for FLBC. For FLBC, the sender can only have a belief about B believing X—not simply about sending some message to B or about B processing some message. Senders do not need to know that message receivers will process a message before sending it. Neither can they know the effects of the message on the receiver. The FLBC system of communication is wholly different than that used by KQML agents.

The standard effects of the FLBC message on B (stated in the language used by KQML) are `considerForKB ¬(bel(A, X))`, `considerForKB bel(A, bel(B, time(before(now), bel(A, X))))` and `considerForKB intend(A, ¬(bel(B, X)))`. These approximately state that B should consider believing that 1) A does not believe that X, 2) A believes that B believes that, up until now, A believed that X, and 3) A wants B to no longer believe that X. This third effect is how an FLBC system approaches the problem of getting another agent to not believe something. The FLBC agent does not advertise that it will believe some future assertion or denial by some other agent. One agent can only hope to convey to another that it wants that agent to believe (or, as in this case, *not* believe) something.

---

delete-one [2, p. 49, 94]

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

Description "A states to B that A wants B to remove the content from its knowledge store."

Formal description "`want(A, ¬(bel(B, X)))`"

Preconditions

A "`know(A, intend(B, process(B, M)))` where M is `delete(A, B, X)`"

B "intend(B, process, B, M)) ∧ bel(B, X)"

Postconditions

A "`know(A, ¬(bel(B, X)))`"

B "`¬(bel(B, X))`"

FLBC

```
<simpleAct speaker="A" hearer="B">
  <illocAct force="dispute">
    <predSt>
      X
    </predSt>
  </illocAct></simpleAct>
```

Informal FLBC A disputes to B that X.

Standard effects

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

Translation of standard effects on B A wants B to believe that 1) A does not believe that X, 2) A believes that B believes that X, and 3) A wants B to not believe that X.

Required vocabulary None.

There seems to be a minor error in the preconditions for this performative: `delete/3` is not a defined performative. I believe it should be `delete-one/3`.

Labrou points out "that it is not *necessarily* the case that `¬(bel(A, X))`. [2, p. 94] One of the standard effects in FLBC of `dispute` is a requirement that the hearer consider believing `¬(bel(A, X))`. Thus, though an FLBC-speaker would not have to disbelieve `(bel(A, X))`, the hearer presumes that this is the case if the hearer believes the speaker.

---

delete-all [2, p. 49+, 94]

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

Description "A wants B to remove the content from its knowledge store."

Formal description "`want(A, ¬(bel(B, X))`"

Preconditions

A "`know(A, intend(B, process(B, M)))` where `M` is `delete(A, B, X)`"

B "`intend(B, process(B, M)) ∧ bel(B, X)`"

Postconditions

A "`know(A, ¬(bel(B, X)))`"

B "`¬(bel(B, X))`"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="dispute">
  <predSt>
   X
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC `A` disputes to `B` that `X`.

Standard effects

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

Translation of standard effects on `B` `A` wants `B` to believe that 1) `A` does not believe that `X`, 2) `A` believes that `B` believes that `X`, and 3) `A` wants `B` to not believe that `X`.

Required vocabulary None.

Again, it seems to me here that the `delete` in the preconditions should be `delete-all`.

Labrou correctly points out that `delete-one` and `delete-all` mean the same thing but that the latter's content has to have at least one free variable. [2, p. 94] That these two performatives mean the same thing is reflected in the FLBC translations; however, the reason for having these two performatives is not clear. The receiving agent certainly can determine if the content has a free variable. If it does, then delete all of them; if it does not, then delete that one term.

---

undelete [2, p. 51, 95]

KQML `(undelete :content <expression>)`

Description "A wants B to restore X in its knowledge store, where X was removed from B's knowledge store upon A's request in the past."

Formal description "`want(A, bel(B, X))`"

Preconditions

A "`know(A, process(B, M))` where `M` is `delete(A, B, X)`"

B "`process(B, M) ∧ ¬(bel(B, X))`"

Postconditions

A "`know(A, bel(B, X))`"

B "`bel(B, X)`"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="report">
  <predSt>
   time(before(now),
    sendMsg(A, B, dispute, X))
  </predSt>
 </illocAct>
 <illocAct force="request">
  <predSt>
   undo(B, sendMsg(A, B,
    dispute, X))
  </predSt>
 </illocAct>
 <illocAct force="assert">
  <predSt>
   X
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC `A` reports to `B` that `A` has previously disputed `X` with `B`. `A` now retracts the disputing of `X` and asserts to `B` that `X`.

Standard effects

```
considerForKB A believes
  time(before(now),
    sendMsg(A, B, dispute, X))
considerForKB A wants (B believes
  time(before(now),
    sendMsg(A, B, dispute, X))
considerForKB A wants do(B,
  undo(B, sendMsg(A, B, dispute, X)))
considerForKB A wants (B wants do(B,
  undo(B, sendMsg(A, B, dispute, X))))
considerForKB A believes X
considerForKB A wants (B believes X)
```

Translation of standard effects on `B` `A` wants `B` to believe that 1) `A` believes that at some time in the past `A` disputed to `B` that `X`, 2) `A` wants `B` to believe that at some time in the past `A` disputed to `B` that `X`, 3) `A` wants `B` to undo the effects of the previously-sent `dispute` message, 4) `A` wants `B` to want to undo the effects of `A` sending `B` the dispute message, 5) `A` believes `X`, and 6) `A` wants `B` to believe that `X`.

Required vocabulary `sendMsg/4`, `undo/2`

Again: there isn't a `delete` performative. Here it should be replaced by `delete-all` ∨ `delete-one`.

The FLBC message is not equivalent to the KQML performative in its preconditions but its effects are very similar.

---

achieve [2, p. 51–52, 95]

KQML `(achieve`
`:content <expression>)`

Description "A wants B to take some action that will make B [sic] true in its VKB. . . ." This is an editing mistake. It should say ". . . make X true . . . ".

Formal description "`want(A, bel(B, X))`. `bel(B, X)` can only be the result of some action outside A and presumably X is a representation of some entity or action in the real world."

Preconditions

A "`know(A, intend(B, process(B, M)))` where `M` is `achieve(A, B, X)`"

B "`intend(B, process(B, M))` ∧ ¬`(bel(B, X))`"

Postconditions

A "`know(A, bel(B, X))`"

B "`bel(B, X)`"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="request">
  <predSt>
   do(B, X)
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC `A` requests of `B` that `B` do `X`.

Standard effects

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

Translation of standard effects on `B` `A` wants `B` to believe that 1) `A` wants `B` to do `X`, and 2) `A` wants `B` to want to do `X`.

Required vocabulary `do/2`.

The FLBC message clearly reflects the meaning of the KQML performative: the speaker wants the hearer to do something.

---

unachieve [2, p. 52, 96]

KQML `(unachieve`
`:content <expression>)`

Description "A wants to undo the effects of an `achieve` that A sent to B in the past.".

Formal description "`want(A, ¬(bel(B, X)))`. `bel(B, X)` can only be the result of some action outside `A` and presumably `X` is a representation of some entity or action in the real world."

Preconditions

A "`know(A, process(B, M))` where `M` is `achieve(A, B, X)`."

B "`process(B, M))` ∧ `bel(B, X)`"

Postconditions

A "`know(A, ¬(bel(B, X)))`"

B "`¬(bel(B, X))`"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="request">
  <predSt>
   undo(B, X)
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC `A` requests of `B` that `B` undo `X`.

Standard effects

```
considerForKB A wants undo(B, X)
considerForKB A wants (B wants
  undo(B, X))
```

Translation of standard effects on `B` `A` wants `B` to believe that 1) `A` wants `B` to undo `X`, and 2) `A` wants `B` to want to undo `X`.

Required vocabulary `undo/2`.

This is the opposite of the `achieve` performative but it is still a request. The FLBC message reflects the limits of this change: it changed in content but not in general attitude (i.e., illocutionary force).

---

advertise [2, p. 52+, 96–97]

KQML `(advertise :content <expression>)`

Description "A wants B to know that A can and will process a message like M from B, if it receives one (A commits himself to a course of action).".

Formal description "`want(A, know(B, intend(A, process(A, M))))`, where `M` is a KQML message `performative_name(A, B, X)`." I believe this last performative should be from `B` to `A` and not *vice versa* as shown. `A` wants `B` to know that `A` intends to process a message, so it must be `A` receiving the message, so it must be a message from `B` to `A`.

Preconditions

A "`intend(A, process(A, M))`"

B "none"

Postconditions

A "`know(A, know(B, intend(A, process(A, M))))`"

B "`know(B, intend(A, process(A, M)))`"

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="promise">
  <predSt>
   do(A, process(B, X))
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC `A` promises `B` that `A` will process a message `X` from `B`.

Standard effects

```
considerForKB A believes (A obligated
  do(A, X)
considerForKB A wants do(A, X)
considerForKB B believes
  (A obligated do(A, X)
```

Translation of standard effects on `B` `A` wants `B` to believe that 1) `A` believes that `A` is obligated to do `X`, 2) `A` wants to do `X`, and 3) `B` believes that `A` is obligated to do `X`.

Required vocabulary `do/2`, `process/2`.

The informal description refers to A committing itself to something. This commitment is reflected in the FLBC message in the illocutionary force of promise. The effects of a promise are that the speaker and hearer both believe the speaker is obligated to an action because of the statement.

---

subscribe [2, p. 54+, 97]

KQML
```
(subscribe
    :content <expression>)
```

Description "A wants to be updated every time that the *response* to the message M is different to [sic] the previous such response.".

Formal description "The semantics of this message are those of M, but additionally every time that B responds to M, it is as if M is sent again and B processes M but delivers the *response* if and only if it is different than the previous one."

Preconditions

- **A** know(A, intend(B, process(B, subscribe(A, B, M)))) plus the preconditions of M
- **B** intend(B, process(B, subscribe(A, B, M))) plus the preconditions of M

Postconditions

- **A** Labrou does not discuss these. I assume it is something like know(A, know(B, subscribe(A, X))) (A knows that B knows that A subscribed to X).
- **B** Labrou does not discuss these. I assume it is something like know(B, subscribe(A, X)) (B knows that A subscribed to X).

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="request">
  <predSt>
   do(B, subscribe(A, M))
  </predSt>
 </illocAct></simpleAct>
```

Informal FLBC A requests of B that B subscribe A to M.

Standard effects

```
considerForKB A wants
  do(B, subscribe(A, M))
considerForKB A wants (B wants
  do(B, subscribe(A, M)))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to subscribe A to M, and 2) A wants B to want to subscribe A to M.

Required vocabulary do/2, subscribe/2.

Labrou's specifications for this performative are less detailed than for most others. However, it seems that this performative represents a request from A to B that B set up a subscription to the performative M.

3.2 Intervention and mechanics

"Of the performatives in this category, we treat only error and sorry as speech acts." [2, p. 98] I do the same.

---

error [2, p. 57+, 98]

KQML
```
(error
    :in-reply-to <word1>)
```

Description "A states to B that A did not process the KQML message M identified by the :reply-with value [X]."

Formal description "¬(process(A, M))"

Preconditions

- **A** "receive(A, M) ∧ ¬(process(A, M))"
- **B** "sendMSG(B, A, M)"

Postconditions

- **A** "know(A, know(B, ¬(process(A, M)))) ∧ ¬(Post_M(A) where Post_M(A) is Post(A) for message M."
- **B** "know(B, ¬(process(A, M))) ∧ ¬(Post_M(B))"

FLBC

```
<simpleAct speaker="A" hearer="B">
  <illocAct force="report">
    <predSt>
      time(T, not process(A, M))
    </predSt>
  </illocAct></simpleAct>
<context respondingTo="RW"></context>
```

Informal FLBC A reports to B that it is not the case that A processed message M at time T. This is in response to message RW.

Standard effects

```
considerForKB A believes time(T,
  not process(A, M))
considerForKB A wants (B believes
  time(T, not process(A, M)))
```

Translation of standard effects on B A wants B to believe that 1) A believes that at time T it is not the case that A processed M, and 2) A wants B to believe that at time T it is not the case that A processed M.

Required vocabulary not/1, process/2

Labrou defines sendMSG(A, B, M) as the state of "A sending message M to B." [2, p. 98] Labrou's discussion of this performative indicate that not being able to process a performative (a precondition for A) can be attributed to either not being able to parse M or M not being an appropriate performative in the flow of conversation. Though I use Labrou's definition, it seems that the originator of M would find this distinction useful.

---

sorry [2, p. 59, 98–99]

KQML (sorry
:in-reply-to <word1>)

Description "A states to B that although it processed the message, it has no (possibly further) response to provide to the KQML message M identified by the :reply-with value [X]."

Formal description "process(A, M)"

Preconditions

- **A** "process(A, M)"
- **B** "sendMSG(B, A, M)"

Postconditions

- **A** "know(A, know(B, process(A, M))) $\wedge$ $\neg$(Post$_M$(A))"
- **B** "know(B, (process(A, M)) $\wedge$ $\neg$(Post$_M$(B))"

FLBC

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

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

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

SAM: continue here

The FLBC translation is exactly that of the eos performative. Both are indicating that there are not going to be any more messages sent in response to an originating message. The difference lies in that with eos the receiver will know that the sender had no more messages to send while with `sorry` the receiver will not be able to make this inference. If this distinction is important to make, then a new predicate, say `finished/1`, could be defined.

The required vocabulary item does not list `respondingTo` because it is already part of FLBC's vocabulary. This is the same reason that `is` is not included for later performatives.

3.3 Facilitation and networking

Labrou asserts that these performatives "are not speech acts in the pure sense." [2, p. 66] It is not clear what he means by this. Performatives in this category help agents find other agents that can process their queries or that can otherwise contribute to responding to a particular message it wants to send.

---

register [2, p. 66, 102]

KQML
```
(register
   :content <word1>)
```

Description "This performative is just a shorthand for (and thus has the same semantic description as) `insert(A, B, X)`, where `X` is a list of properties of agent A, such as its physical address, e-mail address, symbolic name, network protocols that it can process, available proxy agents for `A`, etc."

Formal description `want(A, bel(B, X))`

Precondition

A `know(A, intend(B, process(B, register(A, B, X))))`

B `intend(B, process(B, register(A, B, X)))` $\wedge$ `¬(bel(B, X))`

Postcondition

A `know(A, bel(B, X))`

B `bel(B, X)`

FLBC

```
<simpleAct speaker="A" hearer="B">
  <illocAct force="request">
    <predSt>
      register(A, B)
    </predSt>
  </illocAct>
  <illocAct force="describe">
    <predSt>
      is(A, X)
    </predSt>
  </illocAct>
</simpleAct>
```

Informal FLBC `A` requests of `B` that `A` be registered with `B`. `A` then describes itself to `B` as having features `X`.

Standard effects

```
considerForKB A wants
  do(B, register(A, B))
considerForKB A wants (B wants
  do(B, register(A, B)))
considerForKB A believes is(A, X)
considerForKB A wants
  (B believes is(A, X))
```

Translation of standard effects on `B` `A` wants `B` to believe that 1) `A` wants `B` to register `A` with `B`, 2) `A` wants `B` to want to register `A` with `B`, 3) `A` believes `A` is `X`, and 4) `A` wants `B` to believe that `A` is `X`.

Required vocabulary `register/2`

Labrou describes this performative as being *one* action; however, it seems to be two separate actions. The first is the speaker requesting that it be registered with the addressee. The second is the speaker providing useful information about itself to the addressee. Given that this performative represents these two acts, it is translated as a `request` and a `describe` FLBC message. These assumptions differs from Labrou's assumptions but I believe it provides a clearer (and, as shown later, a more useful) representation.

---

unregister [2, p. 66, 102]

KQML (unregister
:sender A :receiver B)

Description "This is a shorthand for uninsert(A, B, X)), where X is the :content of the register(A, B, X) that A has been sent from the *same* physical address."

Formal description want(A, ¬(bel(B, X)))

Precondition

A know(A, process(B, M)) where M is register(A, B, X)

B process(B, M)) ∧ bel(B, X)

Postcondition

A know(A, ¬(bel(B, X))

B ¬(bel(B, X))

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="retract">
  <predSt>
   is(A, X)
  </predSt>
 </illocAct>
 <illocAct force="request">
  <predSt>
   undo(B, register(A, B))
  </predSt>
 </illocAct>
</simpleAct>
```

Informal FLBC A retracts from B its description. A requests of B that B undo the effects of A registering with B.

Standard effects

```
considerForKB A not believes is(A, X)
considerForKB A believes (B believes
  time(before(now),
    A believes is(A, X)))
considerForKB A wants
  (B not believes is(A, X))
considerForKB A wants do(B,
  undo(B, register(A, B)))
considerForKB A wants (B wants do(B,
  undo(B, register(A, B))))
```

Translation of standard effects on B A wants B to believe that 1) A does not believe that A is X, 2) A believes that B has believed that A is X, 3) A wants B to not believe that A is X, 4) A wants B to undo A registering with B, and 5) A wants B to want to undo B registering A with B.

Required vocabulary undo/2, register/2

Here is where the separation of the register performative into two separate actions makes itself known. Labrou describes this performative as being one action. The FLBC message is two and they correspond to the two from the translation of the register performative. The FLBC message first retracts the previous description that was given and then requests that the previous register be undone.

---

transport-address [2, p. 67, 102–103]

KQML (transport-address
:in-reply-to <word>)

Description "Let us assume that A has sent in the past register(A, B, Y) and that Y is a similar :content to the X in the register [above in this paper]. Then, transport-address signals the relocation of A in the network, to a new physical address and semantically, this act is like a ... unregister(A, B, Y) ... from the old address followed by a ... register(A, B, X) ... from the new address."

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="retract">
  <predSt>
   is(A, Y)
  </predSt>
 </illocAct>
 <illocAct force="describe">
  <predSt>
   is(A, X)
  </predSt>
 </illocAct>
</simpleAct>
```

Informal FLBC A retracts from B that A is Y. A then describes itself to B as having features X.

Standard effects

```
considerForKB A not believes is(A, Y)
considerForKB A believes (B believes
  time(before(now), A believes is(A, Y))
considerForKB A wants
  (B not believes is(A, Y))
considerForKB A believes is(A, X)
considerForKB A wants
  (B believes is(A, X))
```

Translation of standard effects on `B` `A` wants `B` to believe that 1) `A` does not believe that `A` is `Y`, 2) `A` believes that `B` believed that `A` believes that `A` is `Y`, 3) `A` wants `B` to not believe that `A` is `Y`, 4) `A` believes that `A` is `X`, and 5) `A` wants `B` to believe that `A` is `X`.

Required vocabulary None.

Again the separation of the `register` performative into two separate actions makes itself known. Labrou describes this performative as being two actions. The FLBC message is also two but they are slightly different. The FLBC message first retracts the previous description that was given and then provides a new description—but the speaker does not actually *unregister* from the addressee. At all times it remains registered. The only change is the descriptive information about the sender.

---

forward [2, p. 67–68, 103–104]

KQML
```
(forward
     :from A :to B :sender C :receiver D
     :content
       (performative
          :sender A
          :receiver B
          :content X))
```

Description "This performative is a request from agent `:sender` to agent `:receiver` to deliver a message that originated from agent `:from` to agent `:to`. The receiver of the `forward` might be the `:to` agent, in which case the `:receiver` just processes the message in `:content`. Agent `:receiver` might not be able to deliver the message to agent `:to` in which case it should send a `forward` to some other agent that has a better chance to get the message to the `:to` agent." [2, p. 67] "C wants D to send a `forward` message either to B or to some other agent that can hopefully reach B." [2, p. 104]

Formal description "`want(C, sendMSG(D, forward(A, B, D, B, performative(A, B, X))))`"

Preconditions

A "`want(C, sendMSG(D, forward(A, B, D, B, performative(A, B, X))))` $\wedge$ $\neg$`canDeliver(C, B)`"

B "none"

Postconditions

A "none"

B "`sendMSG(D, forward(A, B, D, B, performative(A, B, X)))` $\vee$ `sendMSG(D, forward(A, B, D, E, performative(A, B, X)))`, if there exists an agent `E` for which `bel(C, canDeliver(E, B))`"

FLBC If `D`$\neq$`B`,

```
<flbcMsg msgID="K">
  <simpleAct speaker="C" hearer="D">
    <illocAct force="request">
      <predSt>
        do(D, forward(B, '<flbcMsg
msgID="F"><simpleAct speaker="A"
hearer="B"><illocAct force="G">...
</illocAct></simpleAct></flbcMsg>'))
      </predSt>
    </illocAct>
  </simpleAct>
  <context forwardedBy="C"></context>
</flbcMsg>
```

If `D`= `B`,

```
<flbcMsg msgID="F">
  <simpleAct speaker="A" hearer="B">
    <illocAct force="G">
      ...
    </illocAct></simpleAct>
```

```
    <context forwardedBy="C...">
    </context>
  </flbcMsg>
```

Informal FLBC If D≠B, C requests of D that D forward to B <flbcMsg ...</flbcMsg>. If D=B, A G's of B that [the content of the inner message].

Standard effects If D≠B

```
considerForKB C wants do(D,
  forward(B,
   '<flbcMsg msgID="F">...</flbcMsg>'))
considerForKB C wants (D wants
  do(D, forward(B,
   '<flbcMsg msgID="F">...</flbcMsg>')))
```

If D=B, the standard effects are not those of the forward message but are those of the message that is being forwarded.

Translation of standard effects on B If D≠B: C wants D to believe that 1) C wants D to forward to B the message '<flbcMsg MsgID="F">...</flbcMsg>', 2) C wants D to want to forward to B the message '<flbcMsg MsgID="F">...</flbcMsg>'.

Required vocabulary do/2, forward/2

Labrou provides one forward performative but gives two definitions for it, depending on whether or not B is the same as D—that is, whether or not the intended recipient of the performative is the same as the actual recipient of the current performative. I have provided two separate FLBC messages. The first is defined for the situation in which the receiver is not the agent who is ultimately supposed to receive the message. This is a request from C to D (C may be equal to A) that D forward to B a string which contains an FLBC message. The FLBC message is converted to a string to emphasize that it is not meant to be interpreted by any of the intermediate agents. They are merely being requested to pass along this pile of bits. The force of the string is undefined because the performative in the :content of the forward performative is undefined.

The second message is defined for the situation in which the receiver is the intended recipient of the message. In this case the string representation is converted to an FLBC message and the forwardedBy term in the context is updated to reflect all the agents who forwarded the message.

---

broadcast [2, p. 71, 104]

KQML
```
(broadcast
   :sender A :receiver B
   :content <performative>)
```

Description "This is semantically equivalent to forward(A, B, A, C, performative(A, C, X)) for every agent C that canDeliver(B, C)."

Formal description want(A, sendMSG(C, forward(A, B, C, B, performative(A, B, X)))), where know(B, canDeliver(B, C))

FLBC

```
<flbcMsg msgID="K">
   <simpleAct speaker="A" hearer="C">
     <illocAct force="request">
       <predSt>
         do(C, forward(all, '<flbcMsg
msgID="F"><simpleAct speaker="A">
<illocAct force="G">... </illocAct>
</simpleAct></flbcMsg>'))
       </predSt>
     </illocAct>
   </simpleAct>
</flbcMsg>
```

Informal FLBC A requests of C that C forward to everyone <flbcMsg...</flbcMsg>.

Standard effects

```
considerForKB A wants
  do(C, forward(all, '<flbcMsg
msgID="F">...</flbcMsg>'))
considerForKB A wants (C wants do(C,
  forward(all,
   '<flbcMsg msgID="F">...</flbcMsg>'))
```

Translation of standard effects on B A wants C to believe that 1) A wants C to forward to all appropriate people the message `'<flbcMsg msgID="F">...</flbcMsg>'`, 2) A wants C to want to forward to all appropriate people the message `'<flbcMsg msgID="F">...</flbcMsg>'`.

Required vocabulary `do/2`, `forward/2`

The major difference between this FLBC message and the first message for `forward` is that the first argument of `forward/2` has been changed to `all` and the `hearer` term of the string has been eliminated. The second change makes the string an invalid XML document; however, it will be made valid when C determines the set of B's that are going to receive the message. At that point, for each B, C can insert `hearer="B"` into the `simpleAct` term, add the `context` and `forwardedBy` terms to the message, and send it.

---

broker-one [2, p. 71–72, 105–106]

KQML
```
(broker-one
   :sender A :receiver B
   :content
      (performative
         :sender A
         :content X))
```

Description "This is a request to find an agent that *can* and *will* process the `:content` ... in the name of the receiver of the `broker-one` (so all responses from the third party will be directed to the broker, i.e., $\texttt{receiver}_{broker-one}$)." [2, p. 72]

Formal description This is a complex three-way communication. Between A and B, A is asking that B forward some information to it, acting as a conduit. Between B and D, the semantics are those of `performative(B, D, X)`. Between A and D the semantics are those of `performative(A, D, X)` except that D does not know that the performative came from A and that A does not know who D is.

FLBC
```
<simpleAct speaker="A" hearer="B">
  <illocAct force="request">
    <andMsg>
      <predSt>
        do(B, forward(one, '<flbcMsg
msgID="F"><simpleAct speaker="B">
<illocAct force="G">...</illocAct>
</simpleAct></flbcMsg>'))
      </predSt>
      <simpleAct speaker="B" hearer="A">
        <illocAct force="inform">
          responseTo(F)
        </illocAct>
      </simpleAct>
    </andMsg>
  </illocAct>
</simpleAct>
```

Informal FLBC A requests of B that B forward to one person `<flbcMsg msgID="F">...</flbcMsg>`. A also requests of B that B inform A of responses to this forwarded message.

Standard effects

```
considerForKB A wants do(B,
  forward(one, '<flbcMsg
    msgID="F">...</flbcMsg>'))
considerForKB A wants (B wants do(B,
  forward(one, '<flbcMsg
    msgID="F">...</flbcMsg>')))
considerForKB A wants do(B,
  flbcMsg(B, A, inform, responseTo(F)))
considerForKB A wants (B wants do(B,
  flbcMsg(B, A, inform, responseTo(F))))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to forward to one person the message `'<flbcMsg msgID="F">...</flbcMsg>'`, 2) A wants B to want to forward to one person the message `'<flbcMsg msgID="F">...</flbcMsg>'`, 3) A wants B to inform A of the response to F, and 4) A wants B to want to inform A of the response to F.

Required vocabulary `do/2`, `forward/2`, `responseTo/1`

The effects of the FLBC message are nearly the same as the KQML message—as near as they can be given the different assumptions underlying these two systems. Between A and B, A is asking that B forward a message, without specifying who, and then reply back to A with the eventual response to that forwarded message. Between B and D (whomever that might end up to be), the semantics are those of the forwarded message. Between A and D, the semantics are those of the forwarded message, except that A does not know who D is until it receives the reply and D does not ever know who A is.

---

broker-all [2, p. 72, 106]

KQML
```
(broker-all
   :sender A :receiver B
   :content
      (performative
         :sender A
         :content <expression>))
```

Description "This performative is like multiple broker-one messages for every agent D, such that canProcess(D, performative(A, B, X))."

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="request">
  <andMsg>
   <predSt>
    do(B, forward(all, '<flbcMsg
msgID="F"><simpleAct speaker="B">
<illocAct force="G">...</illocAct>
</simpleAct></flbcMsg>'))
   </predSt>
   <simpleAct speaker="B" hearer="A">
    <illocAct force="inform">
     responseTo(F)
    </illocAct>
   </simpleAct>
  </andMsg>
 </illocAct>
</simpleAct>
```

Informal FLBC A requests of B that B forward to all the message <flbcMsg msgID="F">...</flbcMsg>. A also requests of B that B inform A of responses to this forwarded message.

Standard effects

```
considerForKB A wants do(B,
  forward(all, '<flbcMsg
    msgID="F">...</flbcMsg>'))
considerForKB A wants (B wants do(B,
  forward(all, '<flbcMsg
    msgID="F">...</flbcMsg>')))
considerForKB A wants do(B,
  flbcMsg(B, A, inform, responseTo(F)))
considerForKB A wants (B wants do(B,
  flbcMsg(B, A, inform, responseTo(F))))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to forward to all appropriate people the message '<flbcMsg msgID="F">...</flbcMsg>', 2) A wants B to want to forward to all appropriate people the message '<flbcMsg msgID="F">...</flbcMsg>', 3) A wants B to inform A of the response to F, and 4) A wants B to want to inform A of the response to F.

Required vocabulary do/2, forward/2, responseTo/1

The semantics of the FLBC message are the same as that of multiple broker-one messages.

---

recommend-one [2, p. 72+, 106]

KQML
```
(recommend-one
   :sender A :receiver B
   :content
      (performative
         :sender A
         :content <expression>))
```

Description "This is a request from A to have B forward to it one relevant advertise message from some agent D that suggests that canProcess(D, performative(A, D, X))...."

FLBC

```
<simpleAct speaker="A" hearer="B">
 <illocAct force="request">
```

```
    <andMsg>
     <predSt>
      do(B, evaluate(bel(B,
        makeValue(J, one,
          truthStatus(canProcess(D,
            performative(A, D, X)),
            true)))))
     </predSt>
     <predSt>
      do(B, sendMsg(B, A, inform,
        bel(B, makeValue(J, one,
          truthStatus(isAbleTo(process(D,
            performative(A, D, X))),
            true))),
        [respondingTo(RW)]))
     </predSt>
    </andMsg>
  </illocAct></simpleAct>
```

Informal FLBC A request of B that B determine one agent that can process the message performative(A, D, X). A also requests of B that B inform A of the identity of that agent.

Standard effects

```
considerForKB A wants
  do(B, evaluate(bel(B,
    makeValue(J, one,
      truthStatus(canProcess(D,
        performative(A, D, X)),
        true)))))
considerForKB A wants (B wants
  do(B, evaluate(bel(B,
    makeValue(J, one,
      truthStatus(canProcess(D,
        performative(A, D, X)),
        true))))))
considerForKB A wants
  do(B, sendMsg(B, A, inform,
    bel(B, makeValue(J, one,
      truthStatus(isAbleTo(process(D,
        performative(A, D, X))),
        true)))))
considerForKB A wants (B wants
  do(B, sendMsg(B, A, inform,
    bel(B, makeValue(J, one,
      truthStatus(isAbleTo(process(D,
        performative(A, D, X))),
        true))))))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to determine what one agent B believes can process performative(A, D, X), 2) A wants B to want to determine what one agent B believes can process performative(A, D, X), 3) A wants B to send a message to A informing it of the one agent B believes can process performative(A, D, X), and 4) A wants B to want to send a message to A informing it of the one agent B believes can process performative(A, D, X).

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

---

recommend-all [2, p. 74, 106]

KQML
```
(recommend-all
  :sender A :receiver B
  :content
     (performative
        :sender A
        :content <expression>))
```

Description "This performative is like recommend-one ... for every agent D that has sent advertise(D, B, Y)."

FLBC

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

```
      </predSt>
    </andMsg>
  </illocAct></simpleAct>
```

Informal FLBC A request of B that B determine all the agents that can process the message performative(A, D, X). A also requests of B that B inform A of the identity of those agents.

Standard effects

```
considerForKB A wants
  do(B, evaluate(bel(B,
    makeValue(J, all,
      truthStatus(canProcess(D,
        performative(A, D, X)),
        true)))))
considerForKB A wants (B wants
  do(B, evaluate(bel(B,
    makeValue(J, all,
      truthStatus(canProcess(D,
        performative(A, D, X)),
        true))))))
considerForKB A wants
  do(B, sendMsg(B, A, inform,
    bel(B, makeValue(J, all,
      truthStatus(isAbleTo(process(D,
        performative(A, D, X))),
        true)))))
considerForKB A wants (B wants
  do(B, sendMsg(B, A, inform,
    bel(B, makeValue(J, all,
      truthStatus(isAbleTo(process(D,
        performative(A, D, X))),
        true))))))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to determine what agents B believes can process performative(A, D, X), 2) A wants B to want to determine what agents B believes can process performative(A, D, X), 3) A wants B to send a message to A informing it of the agents B believes can process performative(A, D, X), and 4) A wants B to want to send a message to A informing it of the agents B believes can process performative(A, D, X).

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

---

recruit-one [2, p. 76-77, 106–107]

KQML
```
(recruit-one
   :sender A :receiver B
   :content
      (performative
         :sender A
         :content <expression>))
```

Description "From a semantics point of view this is the same to [sic] a forward(A, C, A, B, performative(A, C, X)) where C is an agent that at some point has sent a advertise(C, B, performative(B, C, X)...." [2, p. 106] "This performative is like a broker-one but responses will be directed back to the issuer of the recruit-one." [2, p. 76]

FLBC

```
<flbcMsg msgID="K">
  <simpleAct speaker="A" hearer="B">
    <illocAct force="request">
      <andMsg>
        <predSt>
          do(B, evaluate(bel(B,
            makeValue(C, one,
              truthStatus(sendMsg(C,
                  B, promise, do(C,
                    process(C,
                      sendMsg(B, C,
                        G, ...)))),
                true)))))
        </predSt>
        <predSt>
          do(B, forward(C, '<flbcMsg
msgID="F"><simpleAct speaker="A"
hearer="C"><illocAct force="G">...
</illocAct></simpleAct></flbcMsg>'))
        </predSt>
      </andMsg>
    </illocAct>
  <context></context>
</flbcMsg>
```

Informal FLBC A requests of B that 1) B determine the identity of one agent (C) whom B believes has promised to process a message with force G from B to C, and 2) B forward to C a message <flbcMsg ...</flbcMsg>.

Standard effects

```
considerForKB A wants
  do(B, evaluate(bel(B,
    makeValue(C, one,
      truthStatus(sendMsg(C, B,
          promise,
          do(C, process(C,
              sendMsg(B, C, G, ...)))),
        true)))))
considerForKB A wants (B wants
  do(B, evaluate(bel(B,
    makeValue(C, one,
      truthStatus(sendMsg(C, B,
          promise,
          do(C, process(C,
              sendMsg(B, C, G, ...)))),
        true)))))
considerForKB A wants do(B,
  forward(C,
   '<flbcMsg msgID="F">...</flbcMsg>'))
considerForKB A wants (B wants
  do(B, forward(C,
   '<flbcMsg msgID="F">...</flbcMsg>')))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to determine the identity of one agent (C) whom B believes has promised to process a message with force G from B to C, 2) A wants B to want to determine the identity of one agent (C) whom B believes has promised to process a message with force G from B to C, 3) A wants B to forward to C the message '<flbcMsg MsgID="F">...</flbcMsg>', 4) A wants B to want to forward to C the message '<flbcMsg MsgID="F">...</flbcMsg>'.

Required vocabulary do/2, bel/2, evaluate/1, makeValue/3, truthStatus/2, sendMsg/4, forward/2

---

recruit-all [2, p. 77, 107]

KQML

```
(recruit-all
   :sender A :receiver B
   :content
      (performative
         :sender A
         :content <expression>))
```

Description "From a semantics point of view this is the same to [sic] a series of forward(A, C, A, B, performative(A, C, X)) messages to all agents C that at some point have sent a advertise(C, B, performative(B, C, X))."

FLBC

```
<flbcMsg msgID="K">
  <simpleAct speaker="A" hearer="B">
    <illocAct force="request">
      <andMsg>
        <predSt>
         do(B, evaluate(bel(B,
          makeValue(C, all,
           truthStatus(sendMsg(C,
             B, promise, do(C,
              process(C,
               sendMsg(B, C,
                G, ...)))),
            true)))))
        </predSt>
        <predSt>
         do(B, forward(C, '<flbcMsg
msgID="F"><simpleAct speaker="A"
hearer="C"><illocAct force="G">...
</illocAct></simpleAct></flbcMsg>'))
        </predSt>
      </andMsg>
    </illocAct>
  <context></context>
</flbcMsg>
```

Informal FLBC A requests of B that 1) B determine the identity of all the agents (C) whom B believes have promised to process a message with force G from B to C, and 2) B forward to all these agents a message <flbcMsg ...</flbcMsg>.

Standard effects

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

```
    makeValue(C, all,
      truthStatus(sendMsg(C, B,
          promise,
          do(C, process(C,
              sendMsg(B, C, G, ...)))),
        true)))))
considerForKB A wants (B wants
  do(B, evaluate(bel(B,
    makeValue(C, all,
      truthStatus(sendMsg(C, B,
          promise,
          do(C, process(C,
              sendMsg(B, C, G, ...)))),
        true)))))
considerForKB A wants do(B,
  forward(C,
   '<flbcMsg msgID="F">...</flbcMsg>'))
considerForKB A wants (B wants
  do(B, forward(C,
   '<flbcMsg msgID="F">...</flbcMsg>')))
```

Translation of standard effects on B A wants B to believe that 1) A wants B to determine the identity of all the agents (C) whom B believes have promised to process a message with force G from B to C, 2) A wants B to want to determine the identity of all the agents (C) whom B believes have promised to process a message with force G from B to C, 3) A wants B to forward to C the message '<flbcMsg MsgID="F">...</flbcMsg>', 4) A wants B to want to forward to C the message '<flbcMsg MsgID="F">...</flbcMsg>'.

Required vocabulary do/2, bel/2, evaluate/1, makeValue/3, truthStatus/2, sendMsg/4, forward/2

4 Results & analysis

The previous section contains the thirty pre-defined KQML performatives and their approximate FLBC translations. The results 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 pred-

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

	Performatives																								
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
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									■										■	■					

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

icates 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 `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 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 `want`s, 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.