DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-9912

A LIGHTWEIGHT CAPABILITY COMMUNICATION MECHANISM

DAVID ROBERTSON
FLÁVIO S. CORRÊA DA SILVA
JAUME AGUSTÍ
WAMBERTO W. VASCONCELOS

Novembro de 1999

A Lightweight Capability Communication Mechanism *

David Robertson
Division of Informatics
University of Edinburgh, Edinburgh, Scotland
dr@dai.ed.ac.uk

Jaume Agustí
Institut d'Investigació en Intel.ligencia Artificial
Campus UAB 08193 Bellaterra, Barcelona, España
agusti@iiia.csic.es

Flávio S. Corrêa da Silva
Instituto de Matemática e Estatística
Universidade de São Paulo, São Paulo, Brazil
fcs@ime.usp.br

Wamberto W. Vasconcelos
Departamento de Estatística e Computação
Universidade Estadual do Ceará, Ceará, Brazil
wvasconcelos@acm.org

Abstract

A persistent problem in managing the interaction between distributed agents is to be able to coordinate the communication between systems without having continually to ask each system for information about what it can do. One form of coordination is through the use of capability descriptions that are advertised by each agent and managed by a brokering mechanism. The task of the broker (which may be centralised or distributed among the agents) is to accept queries and to hypothesise the means of obtaining answers based only on the capability descriptions. This has the advantage that plans for coordinating answers can be constructed by the broker without having to contact the agents. Brokering, however, is not straightforward because capability descriptions can be complex and may be conditional on interactions with other agents. Brokering must also take into account the possibility that the ontologies used by each agent may differ, so some means of relating the terminology of capabilities of agents is needed. Many sophisticated systems exist for tackling parts of this problem but there have been comparatively few attempts to build lightweight engineering solutions by adapting well established methods. We describe a simple way of implementing a lightweight but powerful brokering mechanism.

1 Introduction

Knowledge sharing on a large scale, between significant numbers of agents, is difficult for at least three reasons. One is that it is difficult to connect the ontology of a given system with those of others. A second is that it is difficult to compare the inference methods used in each system [2, 3, 4]. A third is that in systems with automated interaction between agents it is difficult to know which systems should be asked for the information we require. A partial solution to this third problem is through the use of capability descriptions which describe the main facilities which each agent can deliver, thus allowing an assessment to be made about which agents to consult without having to wake up each agent to do so. An architecture for delivering this kind of ability often involves a brokering system, which has the job of assessing capabilities and suggesting how queries to different agents may be combined to discharge complex capabilities. This is a complex problem at the heart of agent communication and, consequently, it is possible to design heavyweight sophisticated systems for dealing with it (see for example the surveys in [8, 11]). We believe, however, that there is a niche for simpler methods which give reasonably sophisticated forms of capability description using well established techniques which, because of their familiarity, are easier for engineers to pick up. This paper describes a method of describing capabilities and brokering them which is formal (our method uses Horn clauses with the normal semantics of this subset of predicate calculus) but is also lightweight in the sense that we use straightforward engineering methods in a tightly focussed way.

A capabilities brokering mechanism has been focus of our interest partially due to the research project **DECaFf-KB** – Distributed Environment for Cooperation Among Formalisms for Knowledge-based Systems – sponsored by the Consortium British Council/CAPES (Brazil), Grant no. 070/98, and with participation of researchers from Universidade de Sao Paulo, Universidade Estadual do Ceara, University of Edinburgh and Institut d'Investigacio en Intel.ligencia Artificial. The main goal of that project is to provide computational means for heterogeneous knowledge based systems to be put to exchange messages and share their knowledge and capabilities to answer queries. The capabilities taken into account in that project are, therefore, descriptive of expressive power and inference mechanisms of the underlying logical systems of each system, as well as the knowledge they contain and are prepared to offer to different systems.

We present our ideas through an illustrative example. In section 2 we summarise our proposed lightweight capabilities brokering architecture; in section 3 we introduce our example, based on interactions between ecological models for environmental monitoring and problem-solving; in sections 4 to 7 we go through the use of our proposed architecture. Finally, in section 8 we draw some conclusions and address future work.

*Work sponsored by the Consortium British Council/CAPES (Brazil), Grant no. 070/98.

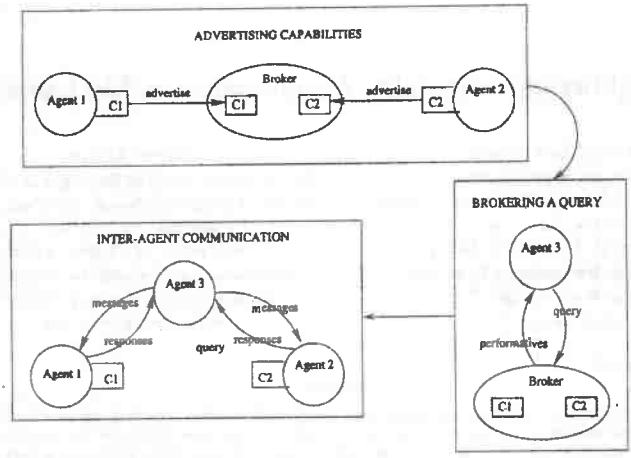


Figure 1: A capability brokering method

2 The Basic Brokering Method

The basic method used in our capability brokering system is depicted in Figure 1. Initially, it is necessary for each participating agent to have described its capabilities in our capability language. In the diagram we depict these capabilities as the boxed C1 and C2 attached to each of the two agents in the example. Agents advertise their capabilities simply by sending these to the broker, which records the capabilities and the agents who claim to be able to supply them. In the next stage, another agent (in our diagram, agent 3) sends a query to the broker. The broker constructs from its capability descriptions (now detached from the agents) its internal description, which we call a "brokerage structure" of how the query might be answered based on those capabilities. It then translates its brokerage structure into a sequence of performative statements describing the messages which it thinks should enable to the query to be satisfied by requesting appropriate agents to discharge their capabilities. In the final stage this performative information is used by the agent which sent the query to select which agents to contact; to send appropriate messages to them; and to await appropriate responses.

The diagram of Figure 1 is only one of the ways in which the method of this paper may be deployed. In particular, the method does not prescribe that the agents submitting queries must be different from the agents supplying capabilities. It also does not prescribe that there must physically be a single, centralised broker to which all the agents advertise and which all queries must be addressed (as in the first stage of the diagram of Figure 1). It would be possible to use our method in a decentralised brokering, provided that each agent had a copy of the brokering mechanism and there was a way of broadcasting capabilities to groups of agents. This substitute to the first diagram of Figure 1 is given in Figure 2.

Decentralised brokers will require additional care in e.g. mantaining consistency among decentralised capabilities descriptions. We are aware of this needed care at implementation stages of decentralised brokers and we intend to develop this issue in future articles. Some interesting techniques do exist to mantain consistency among decentralised pieces of code and information, nevertheless, particularly making use of mobile code. In our specific case, we have considered the possibility of implementing a mobile verifier, that will run cyclically through existing capabilities descriptions collecting and updating information among them. Our idea of mobile verifier has been inspired mostly by [9].

To develop a working instance of this sort of architecture it is necessary to define the following components:

A capability language. This can be understood as a partial specification of the associated agent. It describes key information which can be obtained from the agent without stipulating how it is derived.

A correspondence language. Since the way problems are described and solved may differ between agents there needs to be a way for brokers to spot correspondences between the capabilities of systems. Notice that this is between capabilities only, not between the actual implementations of the agents, which we expect to be much more complex than their capability descriptions. This is an area where deep research is possible but we present here a simple but effective solution.

A capability brokering mechanism. Brokering requires the construction of one or more data structures, each of which which describes a way of supplying the information required by a client based on the capabilities

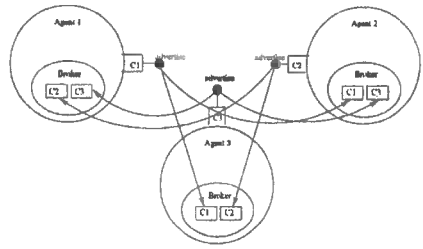


Figure 2: Distributing the brokering mechanism

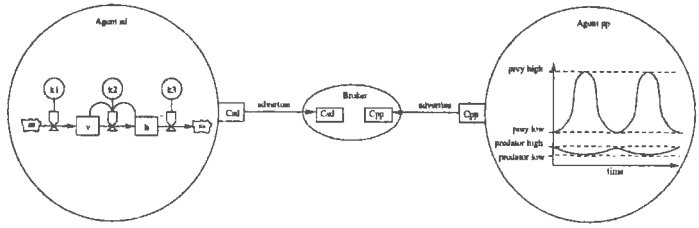


Figure 3: Example of brokering between two modelling systems

advertised by the servers. This must be done automatically because we assume that the client has no innate knowledge of the capabilities of servers or the correspondences between their ontologies.

A translation to performatives. Here we use the word "performative" in deference to the KQML [5] language for message passing between agents. The messages we send can be viewed as simplified forms of KQML performative, although this is not central to our paper. The purpose of performatives in this system is to obtain the information promised from the servers through their capabilities and coordinated in the brokerage structure. The additional conversion is needed, rather than using the brokerage structure directly, because decisions must be made about the ordering of performative messages and these decisions are not uniquely determined by the brokerage structure.

In Sections 4 to 7 we work through each of these components in detail, using a running example for illustration. First, we introduce the example.

3 Running Example

To make our example concrete we describe two agents in detail. These are taken from our earlier work on formal representations of ecological models (see [10]). They are not realistically sized agents but have the advantage of being sophisticated for their size and being based on widely differing approaches to modelling, despite having a certain degree of complementarity. They are therefore good examples for a short paper like this one. An overview of the two models, cast into the brokering framework of Figure 1, is shown in Figure 3. On the left is a model described in System Dynamics notation, describing the dynamics of a herbivore-vegetation system. The boxes are state variables giving the size of the vegetation (`v`) and herbivore (`h`) populations. The arrows represent flows between state variables, which originate and end in an external source-sink (`ss`). These flows are described by equations which are regulated by the parameters (`k1`, `k2` and `k3`). On the right is a model of the cyclic interaction between a predator and prey populations, which we have shown as two synchronised oscillations in their size over time. The most striking feature of these oscillations is that the qualitative sizes of these populations are in opposition - so when the numbers of prey are low the numbers of predators are high and vice-versa. This unusual effect can be caused by time lags in the response of each population to the numbers in the other. In our example, the predator-prey model will describe only this opposition between high and low values, not the quantitative changes as the populations oscillate.

Although the point of our paper is to avoid considering the details of agent implementation when brokering capabilities, it is helpful to consider simple implementations as a way of explaining where distinctions lie. We use the Prolog programming language to describe these implementations because it fits conveniently to the language which we use later to describe capabilities. This is not, however, a fundamental restriction of our method. Our implementations could equally well have been in CLIPS, Lisp or C, provided that we are able to construct an

We have two state variables (v representing vegetation biomass and h representing herbivore biomass) and three parameters (k1, k2 and k3), plus the value for each parameter and the initial value for each state variable.

```
stvar(v). init_value(v, 1000).    param(k1). param_value(k1, 0.01).
stvar(h). init_value(h, 50).      param(k2). param_value(k2, 0.005).
                                  param(k3). param_value(k3, 0.02).
```

We have flows from the source-sink (ss) to v; from v to h; and from h to ss. These are described as difference equations regulated by the parameters.

```
flow(ss, v, Vf, [(k1,K)], Vf is K * 100).
flow(v, h, Hi, [(v,Vp),(k2,K),(h,Hp)], Hi is K * (Vp/Hp)).
flow(h, ss, Ho, [(h,Hp),(k3,K)], Ho is K * Hp).
```

We can find a value for a parameter simply by looking up its value. For state variables, we can do the same at the initial time point but for any later time point we must calculate the value by finding the sum of input and output flows for it and adjusting its previous value by these amounts.

```
value(X, T, V) :- param(X), param_value(X, V).
value(X, T, V) :- stvar(X), init_time(T), init_value(X,V).
value(X, T, V) :- stvar(X), not(init_time(T)), previous(T, Tp),
                  setof(f(I,X,V,A,B), flow(I,X,V,A,B), In),
                  setof(f(X,O,V,A,B), flow(X,O,V,A,B), Out),
                  sum_flows(In, T, SI),
                  sum_flows(Out, T, SO),
                  value(X, Tp, Vp),
                  V is Vp + SI - SO.
```

We can also find out the influence of one variable on another by following through the network of flow connections.

```
influence(X, Y) :- flow(X, Y, _, _, _).
influence(X, Y) :- flow(X, Z, _, _, _), influence(Z, Y).
```

Figure 4: A System Dynamics model

Predators and prey are antagonists of each other. Lynx are predators and Hare are prey.

```
antagonist(X, Y) :- (predator(X), prey(Y)) ; (predator(Y), prey(X)).
predator(lynx).   prey(hare).
```

The qualitative magnitude, Vx, of a population, X, will be the opposite of the qualitative magnitude, Vy, of its antagonist population, Y.

```
size(X, Vx) :- antagonist(X, Y), size(Y, Vy), opposite(Vx, Vy).
```

The only qualitative magnitudes in this model are high and low, which oppose each other.

```
opposite(low, high).   opposite(high, low).
```

Figure 5: An elementary predator-prey model

interface linking the functions in those programs to the capabilities advertised for them. To emphasise that we are making no assumptions about what each agent contains we have made our examples a little non-standard. Figure 4 describes a model of an ecological system in a System Dynamics notation. Figure 5 gives a simple predator-prey model. Although not conventional agent applications, and simplified for the purpose of demonstration, others have conducted knowledge sharing experiments using similar examples - see for instance [1].

To save space, we have not given all of the definitions needed for the systems of Figures 4 and 5. We have, however, enough detail to define some capabilities in the next section.

4 Describing Capabilities

Having defined the two agents of the previous section, we now wish to hide the detail of their knowledge representation language (Prolog) and inference mechanisms (the Prolog interpreter) and we also want to be selective about our claims to other systems about what each agent is able to do. For this we select a convenient language for describing capabilities. This must be constructed by engineers from a variety of backgrounds so its semantics should be as straightforward as possible. It must also be formal because it will be used automatically by the brokering system. We have chosen Horn clauses as our capability language because this provides us with convenient and expressive formal language which, as we shall show, can be used to provide a simple brokering mechanism. We make no claim that this is the best such language.

The intuition behind our use of Horn clauses for capabilities is that we have four forms of capability, C, each of which is defined within the expression $cap(K, C)$, denoting that the agent named K can deliver capability, C in at least one instance or, if not, will signal failure. Valid options for C are:

- A unit goal of the form $P(A_1, \ldots, A_n)$, where P is a predicate name and $A_1, \ldots, A_n$ are its arguments.

- A conjunctive goal of the form $(C_1 \wedge \ldots \wedge C_m)$ where each C_i is a unit goal or a set expression.
- A set expression of the form $setof(X, C, S)$, where C is either a unit goal or a conjunctive goal; X is a tuple of variables appearing in C; and S is a set of instances of those tuples which satisfy C.
- A conditional goal of the form $C_c \leftarrow C_p$, where C_c is a unit goal which the agent, K, will attempt to satisfy (but will not guarantee to satisfy) if the condition, C_p, is satisfied. C_p is either a unit goal or a conjunctive goal.

We can now use this style of capability description for our examples in Figures 4 and 5 (above). The choice of which capabilities to describe requires ingenuity on the part of the engineer of each agent. There is, however, a simple procedure for deciding initially what will be described. We describe this using the System Dynamics agent (named *sd* below):

- First choose the predicate names that correspond to information that might usefully be transmitted by the system. In *sd* we could imagine transmitting information about the values and influences in the system so the predicate names are *value* and *influence*, assuming for convenience a direct translation between the predicate names internal to the agent and those we use in capability descriptions.
- Then decide whether we can provide this information directly or if we need to impose conditions on provision. If we impose conditions then we should start again from the item above for those predicates which appear in the conditions. Information about *influence* can be given directly but for *value* we need to check that we are asking for an appropriate time and state variable. Therefore we add *time* and *stvar* to those which may be transmitted by the system.
- For each of these predicates decide how the information about it is to be transmitted. This can be in one or more of the four forms of capability described earlier. The information on *value*, we already know, is conditional on *time* and *stvar*. The information on *time* can be provided unconditionally but only for individual instances, since the set of possible times is not bounded. The information for *stvar* can be made available either as instances or as the full set because there are only two of these. Continuing this process for all the predicates, and identifying variables appropriately, yields the set of capabilities shown below.

```
cap(sd,  (value(X, T, V) ← time(T) ∧ stvar(X)))
cap(sd,  time(T))
cap(sd,  stvar(X))
cap(sd,  setof(X, stvar(X), S))
cap(sd,  influence(X, Y))
cap(sd,  (stvar(X), setof(Y, (stvar(Y) ∧ influence(X, Y)), S)))
cap(sd,  (stvar(Y), setof(X, (stvar(X) ∧ influence(X, Y)), S)))
cap(sd,  setof((X, Y), (stvar(X) ∧ stvar(Y) ∧ influence(X, Y)), S))
cap(sd,  (value(X, T, V) ← time(T), influence(X, Y)))
cap(sd,  (value(Y, T, V) ← time(T), influence(X, Y)))
cap(pp,  predator(X))
cap(pp,  prey(Y))
```

Notice that for the predator-prey system of Figure 5 we have only two capabilities (labelled *pp* above) which allow information about the names of the predator and prey to be advertised. We were not able to advertise information about the size of the populations in the predator-prey system because it, alone, is not capable of determining this information. There is only one clause in the Prolog program of Figure 5 for determining *size* and this is conditional on information being available about the *size* of the antagonist population – so to know the size of the predator we need to know the size of the prey and vice-versa, but neither is asserted in the predator-prey system. There needs to be a way of advertising a capability which is conditional on the agent in question receiving additional information from another agent, mediated by the broker. For this, we have a partial capability description, $p\_cap(K, C, E)$, which is identical to our original capability description but with an additional argument, E, containing the capability required from another agent. Using this representation for *size* in our predator-prey system we obtain.

```
p_cap(pp,  (size(X, V1) ← predator(X) ∧ prey(Y)),  size(Y, V2))
p_cap(pp,  (size(Y, V2) ← predator(X) ∧ prey(Y)),  size(X, V1))
```

We now turn to the next representational task before brokering: that of defining correspondences between agents.

5 Describing Correspondences

Partial capabilities like the ones in the previous section cannot be discharged unless we either are lucky enough to have identical capabilities being described in different agents or we have taken the trouble to describe correspondences between capabilities. There are numerous ways of expressing such correspondences and no consensus on the best approach. We could, for example, try to build an interlingua [7] into which capabilities from different agents were translated and out of which we could translate the information needed to satisfy partial capabilities. This, however, makes strong assumptions about the integrity of information transmitted through a shared language. For

simplicity, we have instead used definitions which give direct correspondences between pairs of agent capabilities. Each correspondence is of the form $corr(K_1, C_1, K_2, C_2, G) \leftarrow P$ where C_1 is a capability in agent K_1 which corresponds to capability C_2 in agent K_2 with the constraint G restricting the acceptable substitutions for variables in C_1 and C_2. The precondition, P, is an optional conjunction of other correspondences upon which the main correspondence depends. A set of correspondences between *value* in the *sd* agent and *size* in the *pp* agent appear below. The first of these states that values of less than or equal to 10 in *sd* correspond to *low* sizes in *pp* provided that the state variable over which the value is defined in *sd* corresponds to the prey over which size is defined in *pp*. The last of the correspondences below states that the state variable, *h* (the herbivores), in *sd* corresponds to the prey, *hare*, in *pp*.

```
corr(sd, value(X1, T, V), pp, size(X2, low), (V ≤ 10 ∧ C)) ←
    corr(sd, stvar(X1), pp, predator(X2), C)
corr(sd, value(X1, T, V), pp, size(X2, low), (V ≤ 800 ∧ C)) ←
    corr(sd, stvar(X1), pp, prey(X2), C)
corr(sd, value(X1, T, V), pp, size(X2, high), (V > 10 ∧ C)) ←
    corr(sd, stvar(X1), pp, predator(X2), C)
corr(sd, value(X1, T, V), pp, size(X2, high), (V > 800 ∧ C)) ←
    corr(sd, stvar(X1), pp, prey(X2), C)
corr(sd, stvar(h), pp, prey(hare), true)
```

We now have representations of capabilities and correspondences. Our next step is to define a means of generating brokerage structures.

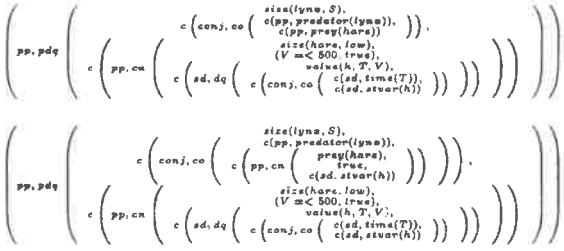
6 A Brokering Mechanism

The purpose of a broker is to find, for a given query posed by a client, the ways in which agents which have advertised their capabilities might be contacted in order to satisfy that query. In our example, the client might want to obtain a prediction of the size of the Lynx population and, to obtain this, it will be necessary to acquire the services of both of our example agents. We need a formal way of representing this sort of combination of capabilities, for which we use what we call a brokerage structure, of the form $c(K, C)$, where K is the name of the agent which should be able to deliver the capability and C is a description of the sources of the capability. C can be in any of the following forms:

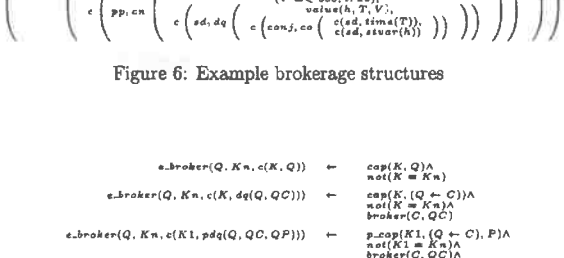
- A capability available directly from K.
- A term of the form $c(K, dq(Q, QC))$, where Q is a capability obtainable from K conditional on its other capabilities and QC describes how these capabilities are obtained.
- A term of the form $c(K, pdq(Q, QC, QP))$, where Q is a capability obtainable from K conditional on its other capabilities and on capabilities external to K, and QC and QP describe how these internal and external capabilities (respectively) are obtained.
- A term of the form $c(conj, co(CQ1, CQ2))$, where $CQ1$ and $CQ2$ are two capability structures which must jointly be satisfied.
- A term of the form $c(K, cn(Q, G, c(K1, Q1)))$, where $K1$ is the name of an agent different from K which allows capability structure Q to be delivered in combination with capability structure $Q1$ provided that the correspondence constraints given by G are satisfiable.

We can now describe a method for constructing brokerage structures of the form given above using the capability and correspondence definitions which reside in the brokering system. Notice that this does not involve any additional interaction with the individual agent systems - the computation can be done entirely within the broker. We describe the algorithm below as a logic program because this is compact, precise and declarative but the mechanism itself could be implemented in a procedural language. The algorithm proceeds by cases corresponding to each of the forms of brokerage structure given above. The *dq* structure is obtained from a conditional *cap* definition; the *pdq* structure from a *p_cap* definition; the *co* structure from two capability structures; and the *cn* structure via correspondence. In all cases where we introduce a new capability into our structure we must demonstrate that it too is obtainable from our definitions - hence the recursive use of *broker* in the algorithm. For partial capabilities (*pdq* structures) we need the same form of brokering but with the constraint that the external capability required by the agent comes from some other source. The easy way to describe this is simply to replicate the *broker* algorithm but with an additional argument (Kn in *e_broker* below) that records the original agent name and prevents it from being used to satisfy the external capability goal.

```
broker(Q, c(K, Q))                    ←  cap(K, Q)
broker(Q, c(K, dq(Q, QC)))            ←  cap(K, (Q ← C))∧
                                         broker(C, QC)
broker(Q, c(K1, pdq(Q, QC, QP)))      ←  p_cap(K1, (Q ← C), P)∧
                                         broker(C, QC)∧
                                         e_broker(P, K1, QP)
broker((Q1, Q2), c(conj, co(CQ1, CQ2)))  ←  broker(Q1, CQ1)∧
                                            broker(Q2, CQ2)
broker(Q2, c(K2, cn(Q2, G, c(K1, BQ))))  ←  corr(K1, Q1, K2, Q2, G)∧
                                            broker(Q1, c(K1, BQ))
```



(1)



(2)

Figure 6: Example brokerage structures

$$
\begin{aligned}
e\_broker(Q, Kn, c(K, Q)) &\leftarrow cap(K, Q) \wedge \\
&\quad not(K = Kn) \\
e\_broker(Q, Kn, c(K, dq(Q, QC))) &\leftarrow cap(K, (Q \leftarrow C)) \wedge \\
&\quad not(K = Kn) \wedge \\
&\quad broker(C, QC) \\
e\_broker(Q, Kn, c(K1, pdq(Q, QC, QP))) &\leftarrow p\_cap(K1, (Q \leftarrow C), P) \wedge \\
&\quad not(K1 = Kn) \wedge \\
&\quad broker(C, QC) \wedge \\
&\quad e\_broker(P, K1, QP) \\
e\_broker((Q1, Q2), Kn, c(conj, co(CQ1, CQ2))) &\leftarrow e\_broker(Q1, Kn, CQ1) \wedge \\
&\quad e\_broker(Q2, Kn, CQ2) \\
e\_broker(Q2, Kn, c(Kn, cn(Q2, G, c(K1, BQ)))) &\leftarrow corr(K1, Q1, Kn, Q2, G) \wedge \\
&\quad broker(Q1, c(K1, BQ))
\end{aligned}
$$

To demonstrate how the algorithm above works we can apply it to our example of brokering a query about the size of the Lynx population. If our broker attempts to satisfy $broker(size(lynx, S), R)$ using the capabilities and correspondences we defined earlier then it can obtain a number of solutions for R. Two of these are given in Figure 6.

The first (expression 1) says that the *pp* agent can answer provided that *lynx* is a predator and *hare* is a prey according to *pp* and that the size of the *hare* population is considered to be low in *pp* via a correspondence to a value of no more than 500 for the biomass of h in the *sd* agent. The *sd* agent should be able to establish this if it is capable of finding an appropriate time and of recognising h as a state variable.

The second brokerage structure (expression 2) is similar to the first except that confirmation that *hare* is a prey within the *pp* agent is not done locally but via a correspondence to the state variable h in the *sd* agent. This raises the issue of preference between brokerage structures. We do not present a solution to this here but there are heuristics for deciding which possible brokerings we would prefer. In our example, the first structure seems preferable to the second because it makes fewer correspondences between expressions in different agents and, since each correspondence raises the possibility of imprecise equivalences having been defined, it seems better to take fewer of these risks.

We are now at the stage where we know what we want to broker but we have not committed to how this might be done. This is the topic of the next section.

7 Assembling a Performative Sequence

The brokerage structures of the previous section describe which pieces of information are worth querying from each agent and stipulate how they can interact in satisfying the query originally posed by a client. They do not, however, prescribe the sequence in which we should transmit messages to the agents which we are coordinating. This is important because, for example, we do not want to ask the *sd* agent for $value(h, T, V)$ until we have asked it for an appropriate time (by asking it to satisfy $time(T)$). How we establish an appropriate sequence of messages depends on the conventions being used for message passing. For the purposes of example, we use a simple convention in which we have three communication acts ("performatives" in the terminology of KQML) that are transmitted sequentially:

- $ask(K, C)$ denoting that we are asking the agent named K to discharge the capability C. We must obtain a response to this message with an instance for C before proceeding with the rest of the sequence.
- $tell(K, C)$ denoting that we are informing the agent named K that a capability which it required externally can be discharged by a correspondence to another agent. We must obtain a response from K indicating that it accepts the information before proceeding with the rest of the sequence.
- $test(G)$ denoting that whatever system is sending the messages should attempt to satisfy the constraint, G, before sending any further messages in the sequence.

We now need an algorithm for translating the brokerage structures of the previous section into message sequences which conform to our message passing conventions. We describe this below in the style of a Definite Clause Grammar (DCG) [6] where an expression of the form $S_h \Rightarrow S$ denotes that we are permitted the sequence represented by the expression S_h if we are permitted the sequence represented by expression S. S can be a sequence (separated by commas) of any of the following expressions: a subsequence expression of the form $S(A_1, \ldots, A_n)$; a call to a predicate of the form $P(A_1, \ldots, A_m)$; or a sequence of terminal symbols of the form $[X_1, \ldots, X_k]$. The grammar is used to generate the sequence of terminal symbols, corresponding to performatives, by unpacking the brokerage structure. We assume in the definitions below that the DCG rules are mutually exclusive, so there is only one possible rule for each form of brokerage subterm. It is readily implemented in Prolog but could be implemented in other languages. Some of the rules require a predicate *dependent_queries*(QC, DQ) which is used to construct the conditions, DQ, for a dependent capability from the brokerage structure, QC. This is a straightforward definition because we simply pull the appropriate subterms out of QC but we omit it here to save space.

```
assemble(c(S, dq(Q, QC)))   ⇒  {dependent_queries(QC, DQ)},
                               assemble(QC),
                               [ask(S, (Q ← DQ))]
assemble(c(S1, pdq(Q, QC, QP)))  ⇒  {dependent_queries(QP, DQ)},
                                    assemble(QC),
                                    assemble(QP),
                                    [ask(S1, (Q ← DQ))]
assemble(c(conj, co(CQ1, CQ2)))  ⇒  assemble(CQ1),
                                    assemble(CQ2)
assemble(c(S, cn(Q, C, CQ)))     ⇒  assemble(CQ),
                                    [test(C), tell(S, Q)]
assemble(c(S, Q))                ⇒  [ask(S, Q)]
```

The assembled message sequences corresponding to brokerage structures 1 and 2 are sequences 3 and 4, respectively, below. Notice how these have flattened the declarative brokerage structures into sequences which respect the procedural realities of message passing. For example, the request (fifth in sequence 3) to the *sd* agent to find a value for h provided that T is a time and h a state variable should come after we have asked whether *sd* is capable of generating a time, T, and of recognising that h is a state variable.

```
[ ask(pp, predator(lynx)),
  ask(pp, prey(hare)),
  ask(sd, time(T)),
  ask(sd, stvar(h)),
  ask(sd, (value(h, T, V) ← time(T) ∧ stvar(h))),
  test((V ≤ 500, true)),
  tell(pp, size(hare, low)),
  ask(pp, (size(lynx, S) ← size(hare, low))) ]                    (3)

[ ask(pp, predator(lynx)),
  ask(sd, stvar(h)),
  tell(pp, prey(hare)),
  ask(sd, time(T)),
  ask(sd, (value(h, T, V) ← time(T) ∧ stvar(h))),
  test((V ≤ 500, true)),
  tell(pp, size(hare, low)),
  ask(pp, (size(lynx, S) ← size(hare, low))) ]                    (4)
```

8 Conclusions

We have described a lightweight but powerful brokering mechanism. It is lightweight because it employs methods which are taught routinely to students and engineers and we might reasonably expect it to be picked up by those groups of people without excessive additional training. It is powerful because it provides an expressive capability language based on predicate logic. Nevertheless, the current system is merely a prototype which needs further development and experimentation. The most immediate points of concern are described below.

- Our current implementation of the algorithm for generating brokerage structures operates in a Prolog-like style, simply attempting to satisfy goals from capabilities using a depth-first search. This has all of the problems which are familiar from normal logic programming, including the potential for non-terminating search. This problem can be addressed by applying well known methods of search control (the simplest of these being to limit the size of brokerage structures).

- In large agent systems the broker may be able to generate huge numbers of brokerage structures by combining the capabilities of different agents. In such cases it is necessary to have a means of choosing preferred structures. There are some obvious heuristics for making this choice, for example by preferring smaller brokerage structures or structures which involve the fewest agents. These are, however, only heuristics and it is not yet clear how effective they would be.

- Our method for describing correspondences by direct association between agents is primitive compared to the forms of knowledge sharing envisaged by others (see Section 5). Because our work is at an early stage when additional problems are a distraction, we have avoided this issue but it seems straightforward to allow the possibility of an interlingua rather than a direct mapping to form correspondences between agent ontologies in our mechanism.

- The language we use for performatives is simplistic compared to systems like KQML. We have yet to explore how brokerage structures may translate to more sophisticated performative languages.
- Our method assumes, as all such systems must, that the connection between advertised capabilities and actual capabilities is sufficiently robust. In other words, if a broker constructs brokerage structures for queries from the capability descriptions of which it has knowledge then the performatives generated from those structures will answer a large number of those queries reliably. We say "a large number" rather than "all" because it is not possible to guarantee reliability in an open system, where we have no idea how carefully each agent was engineered. To improve reliability we need good engineering methods to connect the internal operations of agents to their advertised capabilities.

References

[1] P. Borst, H. Akkermans, and J. Top. Engineering ontologies. *International Journal of Human-Computer Studies*, 46:365–406, 1997.

[2] F.S. Correa da Silva, W.W. Vasconcelos, and D. Robertson. *Cooperation Between Knowledge Based Systems.* 4^{th} World Congress on Expert Systems, 1998, Mexico, pages 819-825.

[3] F.S. Correa da Silva, W.W. Vasconcelos, J. Agusti, D. Robertson, and A.C.V. Melo. *Why Ontologies are not Enough for Knowledge Sharing.* 12^{th} International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (Lecture Notes in Artificial Intelligence, v. 1611), 1999, Cairo (Egypt), pages 520-529, Springer-Verlag.

[4] F.S. Correa da Silva, R.C. Araujo, J. Agusti, and A.C.V. Melo. *Knowledge Sharing Between a Probabilistic Logic and Bayesian Belief Networks.* International Conference on Processing and Management of Uncertainty'2000 (submitted), 2000.

[5] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J.M. Bredshaw, editor, *Software Agents.* AAAI Press/MIT Press, 1997.

[6] G. Gazdar and C. Mellish. *Natural Language Processing in Prolog: An Introduction to Computational Linguistics.* Addison Wesley, 1989. ISBN 0-201-18053-7.

[7] M. Genesereth. Knowledge interchange format. In *Proceedings of the 2nd National Conference on Principles of Knowledge Representation and Reasoning*, pages 599–600. Morgan Kaufmann, 1991.

[8] Y. Labrou, T. Finin, and Y. Peng. Agent communication languages: The current landscape. *IEEE Intelligent Systems*, March/April:45–52, 1999.

[9] V. Nagamuta. *Coordinating Mobile Agents Through a Broadcast Channel.* MSc dissertation, Universidade de Sao Paulo (in portuguese), 1999.

[10] D. Robertson, A. Bundy, R. Muetzelfeldt, M. Haggith, and M Uschold. *Eco-Logic: Logic-Based Approaches to Ecological Modelling.* MIT Press (Logic Programming Series), 1991. ISBN 0-262-18143-6.

[11] G. Wickler. *Using Expressive and Flexible Action Representations to Reason About Capabilities for Intelligent Agent Cooperation.* PhD thesis, Division of Informatics, University of Edinburgh, 1999.

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 1996 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail(mac@ime.usp.br).

Daniela V. Carbogim and Flávio S. Corrêa da Silva
FACTS, ANNOTATIONS, ARGUMENTS AND REASONING
RT-MAC-9601, janeiro de 1996, 22 pp.

Kunio Okuda
REDUÇÃO DE DEPENDÊNCIA PARCIAL E REDUÇÃO DE DEPENDÊNCIA GENERALIZADA
RT-MAC-9602, fevereiro de 1996, 20 pp.

Junior Barrera, Edward R. Dougherty and Nina Sumiko Tomita
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES BY DESIGN OF STATISTICALLY OPTIMAL OPERATORS IN THE CONTEXT OF COMPUTATIONAL LEARNING THEORY.
RT-MAC-9603, abril de 1996, 48 pp.

Junior Barrera e Guillermo Pablo Salas
SET OPERATIONS ON CLOSED INTERVALS AND THEIR APPLICATIONS TO THE AUTOMATIC PROGRAMMINIG OF MMACH'S
RT-MAC-9604, abril de 1995, 66 pp.

Kunio Okuda
CYCLE SHRINKING BY DEPENDENCE REDUCTION
RT-MAC-9605, maio de 1996, 25 pp.

Julio Stern, Fabio Nakano e Marcelo Lauretto
REAL: REAL ATTRIBUTE LEARNING FOR STRATEGIC MARKET OPERATION
RT-MAC-9606, agosto de 1996, 16 pp.

Markus Endler
SISTEMAS OPERACIONAIS DISTRIBUÍDOS: CONCEITOS, EXEMPLOS E TENDÊNCIAS
RT-MAC-9607, agosto de 1996, 120 pp.

Hae Yong Kim
CONSTRUÇÃO RÁPIDA E AUTOMÁTICA DE OPERADORES MORFOLÓGICOS E EFICIENTES PELA APRENDIZAGEM COMPUTACIONAL
RT-MAC-9608, outubro de 1996, 19 pp.

Marcelo Finger
NOTES ON COMPLEX COMBINATORS AND STRUCTURALLY-FREE THEOREM PROVING
RT-MAC-9609, dezembro 1996, 28 pp.

Carlos Eduardo Ferreira, Flávio Keidi Miyazawa e Yoshiko Wakabayashi (eds)
ANAIS DA I OFICINA NACIONAL EM PROBLEMAS DE CORTE E EMPACOTAMENTO
RT-MAC-9610, dezembro de 1996, 65 pp.

Carlos Eduardo Ferreira, C. C. de Souza e Yoshiko Wakabayashi
REARRANGEMENT OF DNA FRAGMENTS: A BRANCH-AND-CUT ALGORITHM
RT-MAC-9701, janeiro de 1997, 24 pp.

Marcelo Finger
NOTES ON THE LOGICAL RECONSTRUCTION OF TEMPORAL DATABASES
RT-MAC-9702, março de 1997, 36 pp.

Flávio S. Corrêa da Silva, Wamberto W. Vasconcelos e David Robertson
COOPERATION BETWEEN KNOWLEDGE BASED SYSTEMS
RT-MAC-9703, abril de 1997, 18 pp.

Junior Barrera, Gerald Jean Francis Banon, Roberto de Alencar Lotufo, Roberto Hirata Junior
MMACH: A MATHEMATICAL MORPHOLOGY TOOLBOX FOR THE KHOROS SYSTEM
RT-MAC-9704, maio de 1997, 67 pp.

Julio Michael Stern e Cibele Dunder
PORTFÓLIOS EFICIENTES INCLUINDO OPÇÕES
RT-MAC-9705, maio de 1997, 29 pp.

Junior Barrera e Ronaldo Fumio Hashimoto
COMPACT REPRESENTATION OF W-OPERATORS
RT-MAC-9706, julho de 1997, 13 pp.

Dilma M. Silva e Markus Endler
CONFIGURAÇÃO DINÂMICA DE SISTEMAS
RT-MAC-9707, agosto de 1997, 35 pp

Kenji Koyama e Routo Terada
AN AUGMENTED FAMILY OF CRYPTOGRAPHIC PARITY CIRCUITS
RT-MAC-9708, setembro de 1997, 15 pp

Routo Terada e Jorge Nakahara Jr.
LINEAR AND DIFFERENTIAL CRYPTANALYSIS OF FEAL-N WITH SWAPPING
RT-MAC-9709, setembro de 1997, 16 pp

Flávio S. Corrêa da Silva e Yara M. Michelacci
MAKING OF AN INTELLIGENT TUTORING SYSTEM (OR METHODOLOGICAL ISSUES OF ARTIFICIAL INTELLIGENCE RESEARCH BY EXAMPLE)
RT-MAC-9710, outubro de 1997, 16 pp.

Marcelo Finger
COMPUTING LIST COMBINATOR SOLUTIONS FOR STRUCTURAL EQUATIONS
RT-MAC-9711, outubro de 1997, 22 pp.

Maria Angela Gurgel and E.M.Rodrigues
THE F-FACTOR PROBLEM
RT-MAC-9712, dezembro de 1997, 22 pp.

Perry R. James, Markus Endler, Marie-Claude Gaudel
DEVELOPMENT OF AN ATOMIC-BROADCAST PROTOCOL USING LOTOS
RT-MAC-9713, dezembro de 1997, 27 pp.

Carlos Eduardo Ferreira and Marko Loparic
A BRANCH-AND-CUT ALGORITHM FOR A VEHICLE ROUTING PROBLEM WITH CAPACITY AND TIME CONSTRAINTS
RT-MAC-9714, dezembro de 1997, 20 pp.

Nami Kobayashi
A HIERARCHY FOR THE RECOGNIZABLE M-SUBSETS
RT-MAC-9715, dezembro de 1997, 47 pp.

Flávio Soares Corrêa da Silva e Daniela Vasconcelos Carbogim
A TWO-SORTED INTERPRETATION FOR ANNOTATED LOGIC
RT-MAC-9801, fevereiro de 1998, 17 pp.

Flávio Soares Corrêa da Silva, Wamberto Weber Vasconcelos, Jaume Agustí, David Robertson e Ana Cristina V. de Melo.
WHY ONTOLOGIES ARE NOT ENOUGH FOR KNOWLEDGE SHARING
RT-MAC-9802, outubro de 1998, 15 pp.

J. C.de Pina e J. Soares
ON THE INTEGER CONE OF THE BASES OF A MATROID
RT-MAC-9803, novembro de 1998, 16 pp.

K. Okuda and S.W.Song
REVISITING HAMILTONIAM DECOMPOSITION OF THE HYPERCUBE
RT-MAC-9804, dezembro 1998, 17pp.

Markus Endler
AGENTES MÓVEIS: UM TUTORIAL
RT-MAC-9805, dezembro 1998, 19pp.

Carlos Alberto de Bragança Pereira, Fabio Nakano e Julio Michael Stern
A DYNAMIC SOFTWARE CERTIFICATION AN VERIFICATION PROCEDURE
RT-MAC-9901, março 1999, 21pp.

Carlos E. Ferreira e Dilma M. Silva
BCC DA USP: UM NOVO CURSO PARA OS DESAFIOS DO NOVO MILÊNIO
RT-MAC-9902, abril 1999, 12pp.

Ronaldo Fumio Hashimoto and Junior Barrera
A SIMPLE ALGORITHM FOR DECOMPOSING CONVEX STRUCTURING ELEMENTS
RT-MAC-9903, abril 1999, 24 pp.

Jorge Euler, Maria do Carmo Noronha e Dilma Menezes da Silva
ESTUDO DE CASO: DESEMPENHO DEFICIENTE DO SISTEMA OPERACIONAL LINUX PARA CARGA MISTA DE APLICAÇÕES.
RT-MAC-9904, maio 1999, 27 pp.

Carlos Humes Junior e Paulo José da Silva e Silva
AN INEXACT CLASSICAL PROXIMAL POINT ALGORITHM VIEWED AS DESCENT METHOD IN THE OPTIMIZATION CASE
RT-MAC-9905, maio 1999, pp.

Carlos Humes Junior and Paulo José da Silva e Silva
STRICT CONVEX REGULARIZATIONS, PROXIMAL POINTS AND AUGMENTED LAGRANGIANS
RT-MAC-9906, maio 1999, 21 pp.

Ronaldo Fumio Hashimoto, Junior Barrera, Carlos Eduardo Ferreira
A COMBINATORIAL OPTIMIZATION TECHNIQUE FOR THE SEQUENTIAL DECOMPOSITION OF EROSIONS AND DILATIONS
RT-MAC-9907, maio 1999, 30 pp.

Carlos Humes Junior and Marcelo Queiroz
ON THE PROJECTED PAIRWISE MULTICOMMODITY FLOW POLYHEDRON
RT-MAC-9908, maio 1999, 18 pp.

Carlos Humes Junior and Marcelo Queiroz
TWO HEURISTICS FOR THE CONTINUOUS CAPACITY AND FLOW ASSIGNMENT GLOBAL OPTIMIZATION
RT-MAC-9909, maio 1999, 32 pp.

Carlos Humes Junior and Paulo José da Silva e Silva
AN INEXACT CLASSICAL PROXIMAL POINT ALGORITHM VIEWED AS A DESCENT METHOD IN THE OPTIMIZATION CASE
RT-MAC-9910, julho 1999, 13 pp.

Markus Endler and Dilma M. Silva and Kunio Okuda
A RELIABLE CONNECTIONLESS PROTOCOL FOR MOBILE CLIENTS
RT-MAC-9911, setembro 1999, 17 pp.

David Robertson, Fávio S. Corrêa da Silva, Jaume Agustí and Wamberto W. Vasconcelos
A LIGHTWEIGHT CAPABILITY COMMUNICATION MECHANISM
RT-MAC-9912, novembro 1999, 14 pp.