Communicative acts and interaction protocols in a distributed information system

Mariusz Nowostawski
Daniel Carter
Stephen Cranefield
Martin Purvis
Department of Information Science
University of Otago
PO Box 56, Dunedin, New Zealand
Phone: (64 3) 479 8083, Fax: (64 3) 479 8311
scranefield@infoscience.otago.ac.nz

Abstract

In FIPA-style multi-agent systems, agents coordinate their activities by sending messages representing particular communicative acts (or performatives). Agent communication languages must strike a balance between simplicity and expressiveness by defining a limited set of communicative act types that fit the communication needs of a wide set of problems. More complex requirements for particular problems must then be handled by defining domain-specific predicates and actions within ontologies. This paper examines the communication needs of a multi-agent distributed information retrieval system and discusses how well these are met by the FIPA ACL.

1 Introduction

Software agents have long been recognised as a promising technology for constructing complex systems as open and distributed communities of loosely-coupled modules [9, 11]. A key development in the field of multi-agent systems has been the specification of agent communication languages (ACLs) such as KQML [3] and FIPA ACL [7]. These languages are intended to provide standard declarative mechanisms for agents to communicate knowledge and make requests of each other.

Agent communication languages must strike a balance between simplicity and expressiveness by defining a limited set of communicative act types (e.g. inform and request) that fit the communication needs of a wide set of problems. More complex requirements for particular problems must then be handled by defining domain-specific actions and predicates within ontologies and by selecting or defining "interaction protocols" (also known as conversation policies)—specifications of the allowed exchanges of messages that may comprise a certain kind of conversation or negotiation.

In the case of the Foundation for Intelligent Physical Agents [7], there is a Communicative Act Library [5] containing a list of standard communicative acts[1] and, for

[1] The FIPA ACL also allows agent developers to use their own non-standard communicative acts.

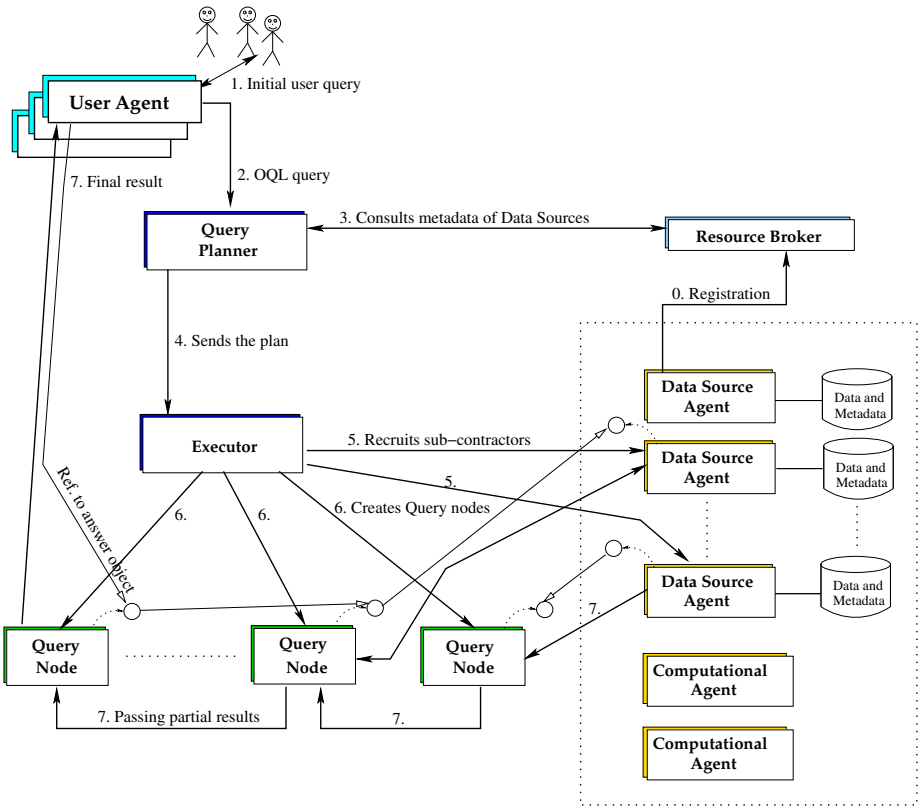

Figure 1: Overview of the NZDIS Architecture

each act, the names and descriptions in English of the allowable message parameters and their meanings, together with specifications in a Belief-Desire-Intention logic of the act's 'feasibility preconditions' and 'rational effect'.

ACL specifications and papers on issues in agent communication tend to present only simple examples of conversations, or do not provide any details on the design decisions involved in expressing a communication scenario using an ACL. In this paper we analyse the relatively complex communication needs of an implemented multi-agent system for distributed information retrieval, discuss the issues involved in encoding the required agent interactions using the FIPA ACL, and assess how well it met our needs.

The structure of the paper is as follows: in Section 2 the overall design and workings of the NZDIS system are presented; in Section 3 we present details of a real-life query which the NZDIS system can perform; this is followed in Section 4 by the details of the ACL communication needed to perform the discussed query; the paper is summarized in Section 5, where conclusions are presented and future work sketched.

2 The NZDIS System

2.1 General overview of NZDIS

The task of integrating disparate data sources is difficult primarily because of the heterogeneity of the available information: data are stored according to widely differing storage formats, media types, and organised according to differing semantics. The challenge is to provide suitable means to integrate such disparate information in a dynamic, open, and distributed environment. The Real Estate New Zealand Distributed Information Systems (NZDIS) [12] research platform is a FIPA-compliant multi-agent framework intended to address this problem.

In NZDIS, information sources are encapsulated as data source agents that accept messages in an agent communication language (FIPA ACL). When a query is entered into the system through a user agent, a number of query processing agents are responsible for discovering from a resource broker agent the data source agents that are relevant to the query, decomposing the query into sub-queries suitable for those agents, executing the subqueries and translating and combining the subquery results into the desired result set. A key component of the query module is the planner agent, which takes a query and breaks it into sub-queries. An executor agent then recruits data source agents and creates new query node agents that each perform one or more of the operations in the plan.

Query results are not returned within an ACL message, but are instead represented by a CORBA object reference which may be used to obtain the result set. This is an example of out-of-band communication: *Agents interact with each other via explicit communicative acts. I.e., we are concerned with explicit models of communication between agents; as opposed to implicit modes of communication as when ants leave pheromone trails for their fellows. It may be that agents have 'out of band' communication amongst themselves. [. . .] However, it is pragmatically expected that out of band communication is limited to semantically neutral communications - such as streaming a video sequence from supplier to customer agents* [8].

A schematic representation of the NZDIS system architecture is shown in Figure 1. All of the solid, directed lines shown in the Figure represent agent messages expressed in FIPA ACL. The circles represent CORBA objects. All the thin lines with white arrow-heads denote references to CORBA objects, and the dotted lines show the creation and ownership of CORBA query result objects. Not shown in this figure is the System Facilitator which maintains a directory of all agents in the system and with which each agent must register when it is incorporated into the system.

2.2 Query Processing subsystem

The user interaction with the user agent is via a specially designed GUI. Currently, the user constructs the query by typing it in as text, with a little help from the visualisation tool, but in the future a fully featured graphical query building process may be provided. A query in Object Query Language (OQL) [1] from the user agent is passed to the Query Planner, which passes it to the Executor, which in turn endeavours to query multiple data sources in order to satisfy the user's query.

The query processing subsystem consists of a planner agent, an executor agent, a resource broker and several data source agents. **The planner agent** generates a query plan involving the retrieval of data from individual data sources and the combination of these data sets using operations such as joins. **The executor agent** takes the plan and

coordinates its execution. **The data source agents** each handle information relating to a single data source. They receive a query encoded using a physical query algebra. The agents transform the query into a form they can handle internally. **The resource broker** has some metadata about each of the data sources (provided by each data source) which it can make available to the planner.

The executor's role in the system is to oversee the distributed processing of the query. As well as coordinating the many external data source agents involved, the executor may spawn one or more short-lived agents that perform some processing of the query. Common examples are buffering and combining operations (e.g. joins).

The executor receives a detailed physical plan in XML format from the planner. The plan details precise operations for the query to be performed. The contents of this XML plan form a graph of connected query or processing nodes. For each node the executor will either instantiate a local agent, or recruit an external data-source or computational agent to perform the operation.

In the NZDIS system, the executor does not act as an intermediary agent collating partial results. Instead results are sent directly to the agent executing the next node in the plan graph. The executor therefore must analyse the dependencies between nodes, and request that an agent executing a node passes the partial result information to the agents responsible for the next nodes in the graph, with whom it may have never communicated with before.

If recruitment of other agents succeeds, then the executor sends a message to all leaf nodes that they may start processing when ready. When an agent finishes processing it must inform the next agent of the results. The result of a query is a collection object modelled after the ODMG 3.0 specification, and Java 2 collection frameworks. The agents pass a reference to the collection in the form of a "stringified" CORBA interoperable object reference (IOR).

3 An example query

In order to illustrate how the system works, we consider an example query: *Given a particular property address, return the area, saleprice, and address, of properties in the same suburb, which have the same land-use*.

We use this query throughout this section to illustrate the process and issues of generating a plan that can be passed on to the executor. After executing the query the executor informs the user agent of a CORBA reference (an IOR) for the object that stores the final result set. We have three data sources available for this query (encapsulated inside data source agents):

- The REDB (Real Estate Database) data source is a Real Estate database that contains information about properties. The back-end is a full object-oriented database, that can handle OQL queries.
- The Geocode service accepts a street address and returns map grid coordinates. It is a lookup service (an example of a computation module), and only works in that single direction – it cannot access coordinates and return an address.
- The LUDB (The Land Use Database) data source comprises some ArcInfo polygon-based information that divides the country into polygons and defines a land-use for each polygon. A somewhat obscure code is used for land-use, for instance "2w 2" refers to a particular sort of flattish high producing land used for intensive

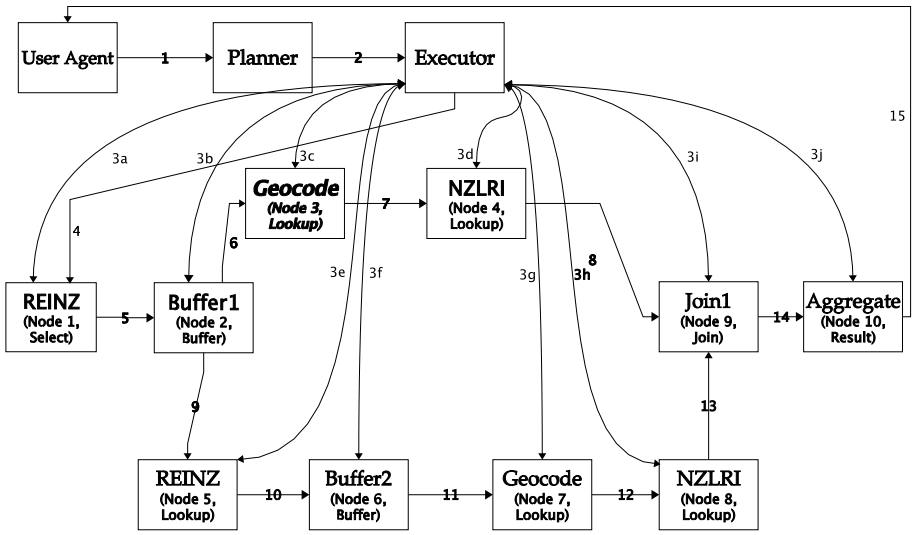

Figure 2: NZDIS Agents Interactions

grazing or forage cropping with some other information too detailed to include here. The LUDB agent takes coordinates and returns a land-use symbol.

The query, for the address “18 Duff Pl, Mosgiel” can be represented in OQL in the following way:

```
select struct(price: p2.SalePrice, area: p2.Area,
              street: p2.StreetNumName)
from Property p1, Property p2
where p1.landInfo.landUse = p2.landInfo.landUse
 and p1.Suburb = p2.Suburb
 and p1.StreetNumName="18 Duff Pl"
```

A plan is generated for this query based on the monoid comprehension calculus [2]. There is still additional work to perform – the plan needs to be unnested and converted into an exact set of steps to be sent to the executor together with a description of the data flow. Details of this process are beyond the scope of this paper. The resulting physical query plan is rather large, so we just discuss a single plan node of it for illustration.

Plan node 7 uses the Geocode data source agent to convert from addresses to coordinates using a semijoin. It takes `p.StreetNumName`, `p.Suburb` and `p.zone.description` found previously in node 5, and looks to match them with `address`, `fine` and `coarse` in the Geocode data so that the coordinates (northing and easting) can be calculated. In addition, the property ID as found previously is passed through to be used later. The inter-agent communication required to setup and execute the full query plan is shown in Figure 2.

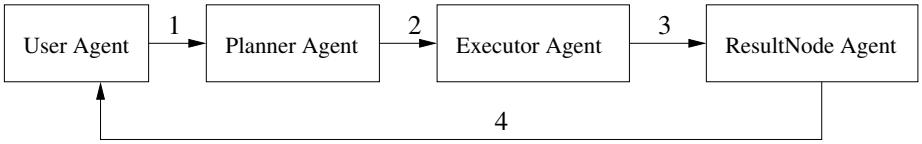

Figure 3: Schematic recruiting in NZDIS system

4 Query processing using FIPA ACL

4.1 Introduction

Examining the FIPA communicative acts and interaction protocols we have found none that directly capture the requirements of the interactions of our system. The fipa-recruiting protocol [6] captures some aspects of the interactions. Like the fipa-brokering protocol [4] it allows for the initiator to make a request to an agent that will not directly fulfil the request, but will know how to locate agents that can. The fipa-recruiting protocol differs in that the agent fulfilling the request will reply directly to the initiator or to some other agent designated by the initiator, instead of passing the reply back through the brokering agent. The brokering and recruiting protocols make use of the `proxy` communicative act, and this is where the misfit to our requirements lies. The proxy communicative act specification states that the proxy message should include an embedded message, which is the message that the broker will send to target agents. In the NZDIS system, no single agent is able to supply an answer to the query, it is thus no good to embed this request inside a proxy message and have the broker forward this message unchanged to other agents. The planner and executor need to translate this request and break it into sub-parts and then recruit many agents to solve individual parts of the query.

In the following sub-sections we detail the way in which we encode the desired coordination between the NZDIS agent using FIPA ACL messages.

4.2 Query execution ontology

An important decision in designing agent interactions is to define the vocabulary used by the agents to express information and requests. This vocabulary is defined by one or more *ontologies* [10]. For the NZDIS system we have a single ontology named *query-execution*. The symbols defined in this ontology are explained below[2].

The user agent uses the following predicate to ask for the results of the query to be made available to it via a CORBA object:

(query-result-ior OQLQueryString IOR)

This is true whenever *IOR* is a reference to a CORBA object containing the result of evaluating *OQLQueryString*. The user agent uses this predicate within a *definite description* [13] that describes the object the user agent wishes to know about as "the thing I that satisfies the proposition *(query-result-ior Q I)*, where Q is the specific query of interest and I is a variable. The message from the final query node to user agent uses it too.

[2]Our ontology does not fully define the meaning of these symbols using a logical description, it just defines their syntax. Providing a full definition would amount to specifying a logical semantics for OQL, which is beyond the scope of this work.

The same definite description is used by the agent that executes the plan's final node to `inform` the user agent that the definite description refers to a particular IOR (in the FIPA SL content language, an equality statement is used to express this).

The planner informs the executor of the generated plan using the predicate:

(query-plan OQLQueryString XMLPlanAsString)

The executor needs to have the notion of plan nodes, identified by string identifiers, and must explicitly reason about when their result sets will be available. To do this it uses the following predicate:

(node-result-ior NodeID IOR)

Finally, the ontology includes an action that the executor asks the agents in charge of the query nodes to perform:

(evaluate-query XMLPlanOperationAsString)

The plan operation given as an argument to this action may refer to the IDs of earlier nodes in the plan. When this action is executed the agent should have already been sent `inform` messages from the agents responsible for those query nodes, containing *node-result-ior* facts specifying the IORs from which those nodes' results can be obtained.

The execution of the *evaluate-query* actions for each query node must be coordinated by the execution agent. The executor does not do this by requesting these actions one by one as each node completes its processing. Instead it uses the `request-when` communicative act to make conditional requests to the node-processing agents, and once they have all agreed to accept these requests the executor sends an `inform` message to the agent responsible for the starting node. These messages contain an assertion that triggers the condition for the earlier *evaluate-query* requests, and the following predicate is used for this:

(query-network-ready PlanID)

where *PlanID* is an identifier assigned by the executor to the plan that the starting node belongs to.

For the non-initial nodes of the plan, the triggering condition for the `request-when` message uses the `node-result-ior` predicate within a conjunction of existential conditions. This conjunction tests whether the agent knows the IORs for the result sets of all earlier plan nodes from which its assigned plan node must obtain data.

4.3 User Agent to Planner

This communication is denoted in Figure 3 as arrow number 1. In plain English, the message is similar to the following utterance: "I would like you to ensure that I am informed about a reference to the object containing the result set for this given query. You may make it happen yourself or, if you need, you can employ all necessary agents to make it happen, and I do not care who will inform me at the end. I just want it to be done.".

This can be done in the following way using the FIPA `proxy` communicative act together with the *fipa-recruiting* interaction protocol:

```
(proxy
  :sender (agent-identifier :name user-agent)
```

```
:receiver (set (agent-identifier :name planner))
:protocol fipa-recruiting
:content
 "((any ?x (delegated-worker (agent-description
                          :name planner) ?x))
    (action (agent-identifier :name planner)
       (request
          (action (agent-identifier :name ?x)
             (inform-ref
                :sender (agent-identifier
                                    :name ?x)
                :receiver (set (agent-identifier
                            :name user-agent))
                :content ((iota ?ior
                     (query-result-ior
                           <QUERY> ?ior)))))))
     true)'')
```

There are some problems with the solution presented above. The above approach misuses the fipa-recruiting protocol, which requires the requested agent, in our case the planner, to proxy the given message unchanged to 3rd parties. In our case however this is not the case, and the planner agent has to tell a harmless "white lie". The planner agrees to the proxy recruitment request, and eventually a 3rd party contacts the user agent with the result. From the user agent's perspective the protocol has been complied with – however the fact that the planner had to do a lot more work and involve many collaborative agents is behind the scenes and invisible to the user agent.

The above solution then does not comply with the FIPA specifications because the receiver of the proxy message does not forward the embedded message, it processes the embedded message and forwards a new message (or set of messages). Thus a solution that uses FIPA communicative acts and protocols as intended will have to take another approach.

What is really needed is more relaxed utterance which allows the planner to modify the original request as much as is needed, to have the requested task done. Assume that we have defined a proprietary action `employ`, which has our required semantic meaning. If we imagine there is a `fipa-task-delegation` ontology with the action `employ` defined there, our message will look as follows:

```
(request
  :sender (agent-identifier :name user-agent)
  :receiver (set (agent-identifier :name planner))
  :protocol fipa-request
  :ontology (set query-execution,fipa-task-delegation)
  :content
    "((action (agent-identifier :name planner)
        (employ
           (inform-ref
             :receiver (set (agent-identifier
                           :name user-agent))
             :content \"((iota ?ior (query-result-ior
                       <QUERY> ?ior)))\"))))")
```

It is possible to have a specialized macro communicative act, with equivalent semantics on the ACL level. This way agents do not need to reference any special ontology for task delegation, and the whole message becomes much simpler.

```
(employ
  :sender (agent-identifier :name user-agent)
  :receiver (set (agent-identifier :name planner))
  :ontology query-execution
  :content
    "((inform-ref
       :receiver (set (agent-identifier
                                :name user-agent))
       :content \"((iota ?ior (query-result-ior
                           <QUERY> ?ior)))\"))")
```

4.4 Planner to Executor

The first message from the planner to the executor is about the plan. The planner simply informs the executor that the plan for a given query has been constructed, and it passes the actual plan encoded in XML to the executor:

```
(inform
  ...
  :language fipa-sl
  :ontology query-execution
  :conversation-id conv01
  :content "((query-plan <QUERY> <PLAN>))"
)
```

The second message from the planner to the executor is similar to the message discussed earlier from the user-agent to the planner. The planner "employs" the executor to have the final result node reference being passed to the user-agent. Note that the message matches exactly the one described in the user-agent to planner case, with the difference that now the final result is not being sent back to the planner, but to the designated-receiver. The designated-receiver is in this case the user-agent.

```
(request
  :sender (agent-identifier :name planner)
  :receiver (set (agent-identifier :name executor))
  :protocol fipa-request
  :ontology (set query-execution,fipa-task-delegation)
  :content
    "((action (agent-identifier :name executor)
        (employ
           (inform-ref
           :receiver (set (agent-identifier
                                :name user-agent))
            :content \"((iota ?ior (node-result-ior
                        result-node ?ior)))\"))))")
```

4.5 Executor to query nodes

The requirements for this speech act are:

1. The executor must ask the data source agent to execute the query parts.
2. The executor must tell the data source to wait until the input data is ready before starting.

3. The results of the query must not come back to the executor, but go to the agent performing the next query node operation.

The FIPA Request communicative act meets the first requirement. The sender is requesting the receiver to perform some action. The content of the message is a description of the action to be performed, in some language the receiver understands. In our case, the executor is requesting the data source agent to evaluate a query. The content is the evaluate-query action taking a query expression argument that the data source agent understands. To meet the second requirement we can look instead to the request-when communicative act. This is similar to request, but adds a precondition. The receiver will not perform the action until the precondition becomes true. The precondition we use is *(exists ?ior2 (node-result-ior <PreviousStepID> ?ior2))* which states the agent must wait until it knows that there exists a node result (IOR) for the previous step on which this query operation depends.

Before query execution can start, we need to ensure all participants have agreed to their requests. There is no point starting execution if it will fail halfway because one agents reject the request. The communicative act `request` does not require that the receiver respond with an agree message. We must then employ an interaction protocol to ensure this. FIPA-request protocol is suitable for this purpose.

In this protocol the executor will play the role of the initiator, and the data source agent will be the participant. For each agent required for the query execution there will be a separate instance of the protocol. Using this protocol then, the data source agent should agree or refuse the request. The executor waits for all participants to agree before it instructs leaf nodes to start processing. If a data source refuses the request then the executor will terminate processing. It cancels the request-when sent to all data agents with a cancel message as per the FIPA-Cancel-Meta-Protocol.

The last phase of the interaction protocol is for the participant to communicate to the initiator inform-done, inform-result, or failure. The problem here is that the result must not be sent to the executor as the initiator, but must be sent to a third agent, being the agent performing the next query operation. The tactic employed here is not to request the results to the query but to request two actions. The first action is to evaluate-query, as defined in the ontology. The second action is to inform the next agent (or agents if this query node is a parallel execution branch) of the query result IOR. This use of the request act, to request the sending of a message is explicitly suggested in the definition of the act. “An important use of the request act is to build composite conversations between agents, where the actions that are the object of the request act are themselves communicative acts such as inform” [5].

The final phase must still be completed to comply with the protocol. After completing the actions, the data source agent will communicate inform-done to the executor. This message is discarded in the current implementation, but could be used to allow the executor to monitor progress of the distributed query execution.

```
(request-when
   :sender (agent-identifier :name executor)
   :receiver (set (agent-identifier
                          :name intermediate-node))
     :language fipa-sl
     :ontology query-execution
     :protocol fipa-request-when
     :content
     "((; (action (agent-identifier
```

```
                    :name intermediate-node)
        (evaluate-query
      <QueryExpressionContainingPreviousStepID>))
       (action (agent-identifier
           :name intermediate-node)
         (inform-ref
           :sender (agent-identifier
                       :name intermediate-node)
           :receiver (set (agent-identifier
                              :name next-node))
           :language fipa-sl
           :conversation-id conv01
           :content
             \"((iota ?ior (node-result-ior
                   intermediate-node ?ior)))\")))
  (exists ?ior2 (node-result-ior <PreviousStepID>
   ?ior2))")
```

As noted earlier the message sent to initial leaf node processing agents differs slightly in that a different precondition is used. Instead of *(exists ?ior2 (node-result-ior <PreviousStepID> ?ior2))* which tests for the existence of a result set ior for the previous query step we would use (query-network-ready <PlanID>) . The executor will inform agents owning the initial nodes that query-network-ready is true if all the node processing agents are ready.

Once the query above has evaluated, the second action will cause this message to be sent to the next node processing agent:

```
(inform
   :sender (agent-identifier
                        :name intermediate-node)
   :receiver (set (agent-identifier
                                :name next-node))
   :language fipa-sl
   :content "((= (iota ?ior (node-result-ior
       intermediate-node ?ior))  <IORString>))")
```

The request from the executor to the root node, that is the final processing node, also looks different. The basic structure is the same, a request-when with two actions, but the inform message that is asked to be sent looks different. This is because the inform will be sent not to another query processing node, but directly back to the user agent. Thus it must appear as an answer to the initiating proxy request from the user agent.

5 Conclusions

In this paper we have examined the communication needs of a multi-agent distributed information retrieval system. Coordinating complex tasks or enterprises in open distributed systems is a very complicated endeavour. FIPA-style communication is constrained by the limited number of predefined performatives, and in some cases must refer to domain specific utterances defined in a custom ontology.

Most of the needs of a real life retrieval system are met by the provided communicative acts from the FIPA communicative acts library. There are some issues only

with anonymous sub-contracting of 3rd parties. When using the proxy communicative act some of the messages become very long with multiple levels of nesting. This makes it very difficult for processing and comprehending even for human developers.

The proposed solution utilises the notion of anonymous subcontracting by means of a custom `employ` action defined in the management ontology defined for the needs of the NZDIS system.

References

[1] R. G. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russel, O. Shadow, T. Stanienda, and F. Velez. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann Publishers, 2000.

[2] L. Fegaras and D. Maier. Optimizing object queries using an effective calculus, 1998. Available online at http://lambda.uta.edu/monoid.ps.gz.

[3] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997. Available at http://www.cs.umbc.edu/kqml/papers/kqmlacl.pdf.

[4] FIPA. FIPA Brokering Interaction Protocol Specification. XC00033G. FIPA specifications online at http://www.fipa.org/specifications/index.html, November 2002.

[5] FIPA. FIPA Communicative Act Library Specification. XC00037I. FIPA specifications online at http://www.fipa.org/specifications/index.html, October 2002.

[6] FIPA. FIPA Recruiting Interaction Protocol Specification. XC00034G. FIPA specifications online at http://www.fipa.org/specifications/index.html, November 2002.

[7] Foundation for Intelligent Physical Agents. FIPA ACL message representation in string specification. http://www.fipa.org/specs/fipa00070/, 2000.

[8] Frank McCabe. Semantics of agent interactions: A white paper. http://www.fipa.org/docs/output/f-out-00125/, 2002.

[9] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7), July 1994.

[10] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):21–66, 1998.

[11] C. J. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, 11(6):24–29, December 1996.

[12] M. Purvis, S. Cranefield, G. Bush, D. Carter, B. McKinlay, M. Nowostawski, and R. Ward. The NZDIS project: an agent-based distributed information systems architecture. In R.H. Sprague Jr., editor, *Proceedings of the Hawaii International Conference on System Sciences (HICSS-33)*. IEEE Computer Society Press (CDROM), 2000.

[13] B. Russell. On denoting. In R. C. Marsh, editor, *Logic and Knowledge: Essays, 1901-1950*. Allen and Unwin, 1956. Also at http://www.santafe.edu/~shalizi/Russell/denoting/.