Referencing Objects in FIPA SL: An Analysis and Proposal

Stephen Cranefield and Martin Purvis
Department of Information Science
University of Otago
PO Box 56, Dunedin, New Zealand
{scranefield,mpurvis}@infoscience.otago.ac.nz

ABSTRACT

Although the syntax and semantics of mainstream agent content languages are based on those of predicate logic, the popularity of the Java programming language, the availability of various free Java-based agent development toolkits and the use of frame-based ontology modelling languages have meant that many developers of multi-agent systems are accustomed to conceptualising their problem domain in terms of classes and objects.

This paper examines the current practice of referring to objects within FIPA SL expressions using functional terms, with particular focus on how well this corresponds to the semantics of the language. After concluding that this use of functional terms is not correct, an extension to the syntax for identifying reference expressions is proposed. This notation allows objects to be referenced in terms of their attribute values in a well defined way, while requiring minimal change to current patterns of use.

1. INTRODUCTION

Agent communication languages (ACLs) such as the Knowledge Query and Manipulation Language (KQML) [6] and the Foundation for Intelligent Physical Agents (FIPA) ACL [7] are based on the idea that "communication can be best modelled as the exchange of declarative statements" [9]. Under this paradigm, agents send, receive and reply to requests for services and information, with the intent of the message specified by a performative (such as 'inform' or 'request') describing the way in which an inner content expression should be interpreted. The content is encoded using a declarative knowledge representation language such as the Knowledge Interchange Format (KIF) [10] or the FIPA Semantic Language (SL) [8]. In addition, the outer ACL expression references one or more 'ontologies' that define the terminology used to denote domain-specific concepts within the message content, and are assumed to be publicly available or at least known to both the sender and receiver agents.

The content languages traditionally used in agent communication, such as KIF and FIPA SL, are based on the formalism of predicate logic. In this style of knowledge representation, domain entities are denoted by terms: constant symbols or compound terms formed from function symbols applied to tuples of terms (the latter are known as *functional terms* in FIPA SL). Elementary facts are represented by atomic statements formed from predicate symbols applied to tuples of terms, and these can be combined into compound statements using connectives such as 'and' and 'or' and quantifiers such as 'for all' and 'there exists'. The content language defines some 'built-in' constants, functions and predicates, and any others used in any given content expression are assumed to be defined in an ontology.

Although the syntax and semantics of mainstream content languages are based on those of predicate logic, the popularity of the Java programming language, the availability of various free Java-based agent development toolkits and the use of frame-based ontology modelling languages have meant that many developers of multi-agent systems are accustomed to conceptualising their problem domain in terms of classes and objects. This observation has led the present authors to investigate the development of object-oriented content languages and tools to support their use in agent platforms [2–5]. Others in the FIPA and Agentcities communities have made use of existing constructs in the FIPA SL language to encode references to objects.

This paper examines the current practice of referring to objects within FIPA SL expressions using functional terms, with particular focus on how well this corresponds to the semantics of the language. After concluding that this use of functional terms is not correct, an extension to the syntax for identifying reference expressions is proposed. This notation allows objects to be referenced in terms of their attribute values in a well defined way.

Although our proposal requires only a minor change in syntax when describing objects, the implications are significant. The current approach in the FIPA and Agentcities communities for referencing objects in terms of their attributes is based on an ad-hoc convention in which the sender's intended meaning of functional terms does not correspond to their logical semantics. This violates a key design principal of high-level agent communication languages such as FIPA ACL: the meaning of messages should be able to be determined based solely on their content, without consideration of any representational shortcuts that the sender may have chosen to make. Our proposal provides a way to retain this fundamental feature of inter-agent messaging while requiring minimal change to current patterns of use.

2. REFERENCING OBJECTS IN FIPA SL

2.1 Using Functional Terms

The FIPA SL specification includes a brief section discussing the syntax of functional terms, and one example presented represents an object as a functional term where the class name is used as a function symbol. This usage has become fairly common in the FIPA community. For example, Willmott et al. [12] make use of a FIPA `inform` message having the following statement as the message content:

```
(= (any ?x (is-car ?x))
   (car
     :colour lightgrey
     :registration "VD 3651"
     :make VW
     :type Golf))
```
(1)

This message has the typical form of an answer to a FIPA `query-ref` message, which might have looked like this:

```
(query-ref
  :sender ...
  :receiver ...                                          (2)
  :language fipa-sl
  :ontology transport
  :content "((any ?x (is-car ?x)))")
```

In this request the agent is asking to know the identity of any entity that satisfies the `is-car` predicate.

In fact, due to some subtleties in FIPA SL's semantics for the `any` operator, statement 1 cannot be used to answer this query because a statement equating an `any` expression with another term or identifying reference expression can never be satisfied[1]. We will therefore rewrite statement 1 in the following form before analysing it further:

```
(member
  (car
    :colour lightgrey
    :registration "VD 3651"                              (3)
    :make VW
    :type Golf)
  (all ?x (is-car ?x)))
```

where `member` is the set membership predicate and the `all` expression is FIPA SL's standard way of referring to the set of all objects that satisfy the given predicate.

In this statement, the inner `car` expression is syntactically a functional term, and this use of a class name as a function symbol is in accordance with the example in the FIPA SL specification. This seems a plausible use of a functional term to represent a domain object because it is likely that there is a unique car object in the intended problem domain with the given attribute values. However, the semantics of functional terms also demand that the following term should have a denotation because functions in standard predicate logic are total:

```
(car
  :colour titaniumsilver
  :registration "VD 3651"
  :make BMW
  :type M3)
```

In this term, the registration number is the same as in the original example, but the other attributes are all different. Presumably the intention of the ontology used for this representation is that the registration number should be a primary key for cars, and so the two `car` objects cannot both exist. However, the semantics of functional terms do not reflect this—both `car` terms above must have a denotation.

As another example, consider the following functional term where `person` is the name of a class modelled in an ontology:

```
(person :name Stephen)
```

Potentially there could be many `person` objects that could be referenced by this expression, as there is no requirement on ontologies that classes' attributes should individually or collectively comprise keys that uniquely identify objects. However, the semantics of functional terms require that every occurrence of a term with the same function symbol and arguments must denote the same entity.

[1] This is due to the use of a Skolem constant in the definition of satisfaction for statements containing `any` expressions. Thanks to an anonymous reviewer for pointing out this problem.

The conclusion that this analysis brings us to is that functional terms are not appropriate for referring to objects, except in the very restricted situation where there exists a unique object for every possible assignment of values to each attribute.

2.2 An Ontology for Describing Objects

Botelho et al. [1] discuss an approach for bridging the gap between object-oriented and propositional representations. In particular, they define two symbols: a predicate `instance` that can be used to state that an entity is an instance of a given class, and a function `value` that takes an object and an attribute name as arguments and returns the value of that attribute for the given object. This can be considered to be a simple ontology for describing objects (although it was not presented as such).

Botelho et al. use this 'ontology' in conjunction with the functional term representation of objects that we criticised above. However, their notation can also be used to avoid the need for this type of functional term. For example, the following could be used within an `inform` message as a more meaningful alternative to statement 3:

```
(and (instance car75 car)
 (and (= (value car75 colour)
         lightgrey)
  (and (= (value car75 registration)
          "VD 3651")
   (and (= (value car75 make) VW)
    (and (= (value car75 type) Golf)
     (member car75
       (all ?x (is-car ?x))))))))
```

2.3 Object-oriented definite descriptions

The notation above provides a way of referencing objects without using functional terms. However, it is, admittedly, rather verbose! In this paper we describe a possible extension to the syntax for definite descriptions in FIPA SL that would allow the flawed statement 3 to be rewritten as follows:

```
(member
  (iota car
    :colour lightgrey
    :registration "VD 3651"
    :make VW
    :type Golf)
  (all ?x (is-car ?x)))
```

The operators `iota`, `any` and `all` are used in FIPA SL to construct "identifying reference expressions" (IREs)—expressions that refer to an entity or set of entities in terms of properties that they satisfy (although the semantics allow for the possibility of there being no objects satisfying those properties). The statement above uses a proposed alternative syntax for an object-oriented IRE. In the following section the current forms of IREs in FIPA SL are described along with an informal description of their semantics compared to functional terms. Following this, a new object-oriented form of IRE is proposed, together with a sketch of how it can be defined in terms of an ontology for describing objects, such as that proposed by Botelho et al.

Note that the statement above answers a `query-ref` (message 2) that used one IRE (the `any` expression) by returning another, more specific, IRE (the `iota` expression). We do not consider this to be a problem, and in fact this seems to be a normal pattern of interaction in human conversation.

3. IDENTIFYING REFERENCE EXPRESSIONS

In principle, the ontologies known to two communicating agents should allow the construction of terms that denote all entities in their shared domain of discourse. However, it cannot be assumed that both agents know the appropriate term to represent every entity of interest[2]. Therefore, it is necessary to provide a way for agents to refer to entities by what *is* known about them, i.e. by describing them in terms of properties that they satisfy. This capability is provided by FIPA SL's identifying reference expressions (IRE).

3.1 Current Syntax and Semantics

In FIPA SL, the `iota` operator can be used to bind a free variable in a proposition, and the resulting expression is taken to be an alias for the (supposedly unique) term that satisfies the proposition when substituted for the bound variable. There are also operators `any` and `all` that refer to some arbitrary, or (respectively) the set of all, entities that satisfy a proposition.

The following is an example of an IRE, where we assume that `father-of` is a predicate relating a person (the first argument) to their biological father (the second argument):

```
(iota ?x (father-of bob ?x))
```

This expression seems well defined at first glance, given that everyone has a unique father. However, this is based on the assumption that the constant `bob` denotes a person rather than (say) a rock, and in an untyped logic this can't be guaranteed simply from a syntactic analysis of the expression. It is therefore possible that this expression does not have a denotation. In this situation, following the work of Bertrand Russell [11], the expression is taken to be meaningless, as is any expression that contains it as a sub-expression.

A stronger demonstration of the potential meaninglessness of IREs is given by the following example:

```
(iota ?x (father-of ?x bob))
```

Even if we assume that `bob` denotes a person, it cannot be guaranteed that the expression has a denotation, because the `iota` operator is intended to refer to a unique entity satisfying the proposition, whereas Bob could have more than one child.

The possible lack of a denotation for an IRE (or at least one formed using `iota` or `any`) is in contrast to the semantics of functional terms, which always have a (unique) denotation in standard predicate logic. For example, it would be possible to include a unary `father` function symbol in an ontology (allowing Bob's father to be denoted by `father(bob)`), but a similar `child` function could only be used if the ontology were intended to model a society that enforced a one-child policy.

3.2 Object-oriented IREs

Section 2.1 discussed how the semantics of functional terms mean that (in most cases) they cannot be used to represent objects. In particular, we need semantics that allow for the possibility that expressions that reference objects in terms of their attribute values may be undefined. This is precisely the issue that the semantics of IREs address (at least those using the `any` and `iota` operators). We therefore propose that instead of using functional terms to refer to objects, FIPA SL should be extended to allow an `iota`, `any` or `all` operator to be followed by a class name and a list of attributes (written using the SL parameter notation) and their associated values, in order to refer to the/any/all object(s) with the given attribute values.

For example (adapting the functional term example from the FIPA SL specification), a particular vehicle might be referred to as follows (note that this example uses a different ontology from the `car` example in Section 2.1):

```
(iota vehicle
  :colour red
  :max-speed 100
  :owner (iota person
           :name Stephen
           :nationality "New Zealander"))
```

Note that there are two objects referred to in this example, and all that is needed to convert this expression from its analogue in current practice (where it would be represented as two nested functional terms) into this form is the addition of two `iota` symbols.

The semantics for this can be defined in terms of a translation into a more complex expression using the existing SL syntax and an ontology for describing objects such as the one described above:

```
(iota ?x
 (and (instance ?x vehicle)
  (and (= (value ?x colour) red)
   (and (= (value ?x max-speed) 100)
    (= (value ?x owner)
       (iota ?y
        (and (instance ?y person)
         (and (= (value ?y name) Stephen)
          (= (value ?y nationality)
             "New Zealander")))))))))
```

4. CONCLUSION

This paper has investigated the use of object-oriented ontologies in conjunction with the use of logic-based content languages. In particular, the current practice of referring to objects within FIPA SL content expressions using functional terms was examined and found not to correspond with the semantics of the language. An extension to FIPA SL's syntax for identifying reference expressions was proposed, which allows objects to be described in terms of their attribute values without the semantic problems associated with current practice. This notation will allow FIPA agent developers to refer to objects in a well defined way while requiring minimal change to their current patterns of use.

5. REFERENCES

[1] L. Botelho, N. Antunes, M. Ebrahim, and P. Ramos. Greeks and Trojans together. In *Proceedings of the Workshop on Ontologies in Agent Systems, 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2002. http://CEUR-WS.org/Vol-66/oas02-8.pdf.

[2] S. Cranefield, M. Nowostawski, and M. Purvis. Implementing agent communication languages directly from UML specifications. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, volume 2, pages 553–554. ACM Press, 2002.

[3] S. Cranefield and M. Purvis. Extending agent messaging to enable OO information exchange. In R. Trappl, editor, *Proceedings of the 2nd International Symposium "From Agent Theory to Agent Implementation" (AT2AI-2) at the 5th*

[2] In fact, it cannot be assumed that the two agents' world models have the same denotation for every term, but exploring this issue is beyond the scope of this paper

European Meeting on Cybernetics and Systems Research (EMCSR 2000), Vienna, 2000. Austrian Society for Cybernetic Studies. Published under the title “Cybernetics and Systems 2000”. An earlier version is available at http://www.otago.ac.nz/informationscience/publctns/complete/papers/dp2000-07.pdf.gz.

[4] S. Cranefield and M. Purvis. A UML profile and mapping for the generation of ontology-specific content languages. *Knowledge Engineering Review*, 17(1):21–39, 2002.

[5] S. Cranefield, M. Purvis, and M. Nowostawski. Is it an ontology or an abstract syntax? – Modelling objects, knowledge and agent messages. In *Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods*, pages 16.1–16.4, 2000. http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/16.pdf.

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

[7] FIPA. FIPA ACL message representation in string specification, Foundation for Intelligent Physical Agents, 2002. http://www.fipa.org/specs/fipa00070/.

[8] FIPA. FIPA SL content language specification, Foundation for Intelligent Physical Agents, 2002. http://www.fipa.org/specs/fipa00008/.

[9] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994.

[10] NCITS. Draft proposed American national standard for Knowledge Interchange Format, National Committee for Information Technology Standards. http://logic.stanford.edu/kif/dpans.html, 1998.

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

[12] S. Willmott, J. Dale, and P. Charlton. Agent communication semantics for open environments. Input Document f-in-00066, Foundation for Intelligent Agents, 2002. http://www.fipa.org/docs/input/f-in-00066/.