Modelling the Communication Process Between Agents

Ofelia Vasilescu

E-mail: vofelia@u3.ici.ro

Abstract: The purpose of this paper is to present a language for the modelling of communication between agents, that includes deontic and illocutionary constructs. Deontic concepts, as obligations, permission, and prohibitions are essential for modelling communication between agents. The description the dynamics of obligations, i.e. how obligations are created and destroyed, uses the tools of the speech act theory. Since it is a logic programming language, it has simple semantics and is executable.

Keywords: cooperating agent, logic programming language, deontic concepts, illocutionary acts, instrumental acts

1. Introduction

Technical advances in computer networking and organizational demands made decentralized and co-operating information systems get ever more important. In order to specify and design these systems, we need the concept of intelligent and cooperating agents. The term "agent" means an entity, human or machine, that functions continuously and autonomously in an environment in which other processes are carried on and other agents exist. Agents are often taken to be "high-level", meaning that they can be described in mental terms such as beliefs, knowledge, capabilities, decisions, plan, goals, desires, intentions, obligations, commitments, etc.

A vast literature was dedicated to different mental aspects. The social relations of obligations, prohibitions, and permissions have been studied in deontic logic, and can be found in [Gabbay, 1984], [Meyer,1993]. One of the first axiomatic systems for deontic logic was the "standard system" KD defined by von Wright in 1951 [Wright, 1951], where the deontic operators O (for Obligation) and P (for Permission) were introduced together with a small number of axioms stating their properties and interrelationships. In 1964, von Wright extended his original system to a system for dyadic logic [Wright, 1964], where conditional obligations were introduced. Meyer suggested in 1988 to reduce deontic logic to a variant of dynamic logic [Meyer,1993]. The dynamic logic led to the strict representation between actions and assertions. An action may change the current situation, an assertion may not. Meyer considered the obligations as pertaining only to actions, not to assertions. Only could he specify the obligations, but there is no tool for the dynamic creation of obligations. In 1995, Dignum et al showed how such constructs could be provided through a combination of deontic logic and illocutionary logic [Dignum, 1995]. Illocutionary logic is a formalisation of the theory of speech acts [Searle79], which can be used for modelling communication structures. One must distinguish [Dignum, 1995] between speech acts and instrumental acts; speech acts are used for communication between agents, whereas instrumental acts are not.

The aspects of obligations and commitments have been largely studied recently. These aspects are essential for cooperation and coordination among agents. When coordinating their activities, agents create and fulfil obligations, that is agents ask each other to do things, they promise to do things, and they carry out what has been requested or promised. Thus each agent holds an agenda describing what it is obliged to do in the future. This agenda is not static, but changes as the agent adds and removes obligations. The agent removes obligations by meeting them. The agent adds new obligations to its agenda as a response to requests from other agents. This means that agents use speech acts (directives, commissives, and declaratives) to create and cancel their obligations. Speech acts, therefore, provide a natural basis for a structured description of actions that take place in agent communication and interaction.

In this paper, we introduce a language based on first order logic for creating, specifying and monitoring obligations between agents. The paper is organised as follows. Section 2 introduces the language and gives its syntax and semantics. Section 3 illustrates a communication process between agents. Finally, Section 4 draws the conclusions.

2. Language of Communication Between Agents

There are two main approaches to designing an agent communication language. The first approach is procedural, where communication is based on an executable content. This could be done using such programming languages as Java. The second approach is declarative, where communication is based on declarative statements, such as definitions and assumptions. Because of the limitations of the procedural approach (e.g. it is difficult to check and co-ordinate an executable content), declarative languages have been preferred in the design of agent communication languages. Most declarative language implementations are based on illocutionary acts, such as requesting or commanding; such actions are commonly called performatives. Two more popular declarative agent languages are: the Knowledge query and manipulation language (KQML) and the knowledge interchange format (KIF).

We introduce a language for specifying, creating, and monitoring obligations based on the first order logic. This approach was inspired by von Benthem et al in [Benthem et al, 1995] where first order logic was used to model dynamics. In order to provide logic programming semantics, the syntax of language is restricted accordingly. This approach has several advantages:

- *Simple semantics*. The meanings of the language constructs are easy to understand as they are given using first order semantics.
- *Executable specifications*. As they can be interpreted as ordinary logic programs, specifications in the language are executable.
- *Explicit creation of obligations*. The language makes it possible to explicitly create obligations by means of speech acts.

2.1 Syntax

The syntax of language is now defined. The alphabet, as usual for first order predicate logic, consists of a set C of constants, a set V of variables, a set F of function symbols, a set P of predicate symbols, a set of connective symbols and a set of punctuation symbols. The Universe of Discourse consists of different kinds of objects including agent, time points, actions and state of affairs. In order to distinguish between terms denoting such different kinds of objects, the terms are typed. We introduce the types Ag for agents, T for time points, A for actions, and SoA for states of affairs. We also introduce the type AS for the contents of speech acts, which may be actions, states of affairs, or combinations of these; A and SoA are subtypes of AS. To enable the construction of different actions and states of affairs, a special set of function symbols is defined:

- a set $Ac = \{\sim, \wedge, \vee\} \subset F$ of connectors;
- a set $IP = \{\mathrm{dir_c}, \mathrm{dir_a}, \mathrm{com}, \mathrm{com\_cond}, \mathrm{auth}, \mathrm{retract}\} \subset F$ of illocutionary points;
- a set $IA \subset F$ of instrumental acts constructors;
- a set $SoA \subset F$ of states of affairs constructors;
- a set $DO = \{ \mathrm{O}, \mathrm{Au}, \mathrm{Pe}, \mathrm{Fo} \} \subset SoA \subset F$ of deontic operators;

These function symbols are typed as follows:

$$\sim : AS \rightarrow AS$$

$$\wedge : AS \times AS \rightarrow AS$$

$$\vee : AS \times AS \rightarrow AS$$

$$\mathrm{dir_c} : Ag \times Ag \times AS \times T \rightarrow SA$$

$$\mathrm{dir_a} : Ag \times Ag \times AS \times T \rightarrow SA$$

$$\mathrm{com} : Ag \times Ag \times AS \times T \rightarrow SA$$

$$\mathrm{com\_cond} : Ag \times Ag \times AS \times T \times AS \times T \rightarrow SA$$

$$\mathrm{auth} : Ag \times Ag \times AS \times T \rightarrow SA$$

$$\mathrm{retract} : Ag \times Ag \times AS \times T \rightarrow SA$$

$$\mathrm{O} : Ag \times AS \times T \times T \rightarrow SoA$$

$$\mathrm{Au} : Ag \times AS \times T \times T \rightarrow SoA$$

$$\mathrm{Pe} : Ag \times AS \times T \times T \rightarrow SoA$$

$$\mathrm{Fo} : Ag \times AS \times T \times T \rightarrow SoA$$

The predicate symbols in this language are $<$, $\leq$, $=$, done and holds with arity two and fulfilled with arity three. The predicate symbols holds, done, and fulfilled are typed as follows:

$$\mathrm{holds} : SoA \times T$$

$$\mathrm{done} : A \times T$$

$$\mathrm{fulfilled} : AS \times T \times T$$

A well set formula in this language is a clause in Clausal Normal Form [Lloyd, 1987], i.e. an ordinary logic programming formula. We use Prolog notation and adopt the convention that constants are denoted by lower-case letters and variables by upper-case letters.

The function symbols introduced above shall be read as follows:

dir_c (Ag1,Ag2,AS,T)- Ag1 asks Ag2 to fulfil AS latest at T;

dir_a (Ag1,Ag2,AS,T)- Ag1 requests that Ag2 shall fulfil AS latest at T;

com (Ag1,Ag2,AS,T) - Ag1 commits itself to Ag2 to fulfil AS latest at T;

com_cond(Ag1,Ag2,AS_1,T1,AS_2,T2) - Ag1 commits itself to Ag2 to fulfil AS_2 latest at T2 if AS_1 is fulfilled at T1;

auth (Ag1,Ag2,AS,T)- Ag1 authorizes Ag2 to request that Ag1 shall fulfil AS before T;

retract (Ag1,Ag2,AS)-Ag1 withdraws the authorization from Ag2 to request that Ag1 shall fulfil AS;

O(Ag,AS,T1,T2)- It is obligatory for Ag to fulfil AS between T1 and T2;

Au(Ag,AS,T1,T2)- Agent Ag is authorised to request that AS shall be fulfilled between T1 and T2;

Pe(Ag,AS,T1,T2)- Agent Ag is permitted to fulfil AS between T1 and T2;

Fo(Ag,AS,T1,T2)- Agent Ag is forbidden to fulfil AS between T1 and T2;

¬AS - The negation of AS;

$AS_1 \wedge AS_2$ - The conjunction of AS_1 and AS_2;

$AS_1 \vee AS_2$ - The disjunction of AS_1 and AS_2;

The predicate built by holds, done, and fulfilled shall be read as follows:

holds(SoA,T)- The state of affairs SoA holds at T;

done(A,T)- The action A has been performed at T;

fulfilled(AS,T1,T2)- AS is fulfilled with respect to T1 and T2, i.e. AS is performed between T1 and T2 (if AS corresponds to an action), or AS holds at T2 (if AS corresponds to a state of affairs).

2.2 Semantics

The well set formulas of the language are clauses in Clausal Normal Form, since the language inherits its semantics from logic programming. Here is a number of axioms in the form of rules for the language constructors.

The meaning of the predicate fulfilled is defined by the following rules:

A1. fulfilled(SoA, T1,T2)←
holds(SoA,T2).

A2. fulfilled(A,T1,T2)←
done(A,T3).
T1 ≤ T3 ≤ T2.

A3. fulfilled($AS_1 \wedge AS_2$,T1,T2)←
fulfilled(AS_1,T1,T2),
fulfilled(AS_2,T1,T2).

A4. fulfilled($AS_1 \vee AS_2$,T1,T2)←
fulfilled(AS_1,T1,T2).

A5. fulfilled($AS_1 \vee AS_2$,T1,T2)←
fulfilled(AS_2,T1,T2).

A6. fulfilled(¬AS,T1,T2)←
¬fulfilled(AS,T1,T2).

A7. holds(O(Ag1,AS,T1,T3),T2)←
done(dir_a(Ag2,Ag1,AS,T3),T1),
holds(Au(Ag2,Ag1,AS,T4,T5),T1),
T1 ≤ T2 ≤ T3,
T4 ≤ T1 ≤ T5.

A8. holds(O(Ag1,AS,T1,T3),T2)←
done(dir_c(Ag2,Ag1,AS,T3),T1),
holds(Pe(Ag2,Ag1,AS,T0,T3),T1),
T0 ≤ T1 ≤ T2 ≤ T3.

A9. holds(O(Ag1,AS,T1,T3),T2)←

done(com(Ag,_,AS,T3),T1),

¬fulfilled(AS,T1,T2),

T1 ≤ T2.

A10. holds(O(Ag1,AS_2,T1,T4),T2)←

done(com_cond(Ag1,Ag2,AS_1,T3,AS_2, T4),T1),

fulfilled(AS_1,T1,T3),

¬fulfilled(AS_2,T3,T4),

T1 ≤ T2 ≤ T3 ≤ T4.

A11. holds(Au(Ag,AS,T1,T3),T2)←

done(auth(_,Ag,AS,T3),T1),

T1 ≤ T2 ≤ T3.

3. Example

Let us consider the case of communication between a customer and an intelligent decision support system viewed like a multi-agent system. Figure 1 presents the communication process between customer and coordination agent.

The customer asks that the co-ordination agent of the system performs a product. Given a product demand, the co-ordination agent accepts the request. That means that if the customer initiates an order for the product of the demand, the co-ordination agent has to fulfil it by delivering the product. After delivery, the customer has to pay for the product. In this example, the actions which may come up in the domain are speech acts and instrumental acts. Product_demand, accept_demand and order correspond to speech acts and deliver and pay are instrumental acts. For these actions we define the corresponding function symbols, as follows:

{product_demand, accept_demand,order} $\subset SA$ and {deliver, pay} $\subset IA$.

The action that a customer asks the coordination agent to perform a product P in a quantity Q latest at Tf is in effect a clear directive of the customer for the co-ordination agent that the co-ordination agent shall submit a request by accept_demand. This is formalised as domain axiom D1.

D1.
done(dir_c(Customer,AgC,accept_demand(AgC,Customer,P,Q,Tf),T2),T1)←

done(product_demand(Customer,AgC,P,Q,Tf),T1).

The demand is accepted if the coordination agent knows the specification of product asked by the customer and if it can produce it until Tf:

D2.
done(accept_demand(AgC,Customer,P,Q,Tf),T)←

done(product_specification(AgC,Cu

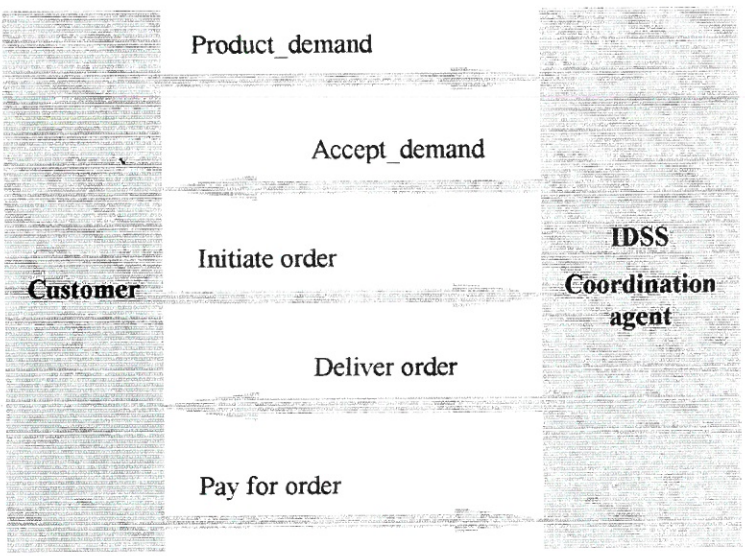

Figure 1. Ordering Process

stomer,P,Q,Tb,Te),T),

Te ≤ Tf.

The product_specification action is also an authorisation, where the co-ordination agent authorises a customer for a delivery, i.e. if the customer requests a delivery then the co-ordination agent should comply to. This is expressed in the following axiom:

D3. done(auth(AgC,Customer,deliver(AgC,

Customer,P,Q,Te),Tb),T)←

done(product_specification(AgC,

Customer,P,Q,Tb,Te),T).

The action that the customer gives an order to the co-ordination agent after having received the accepted demand is an authority directive, expressed in axiom D4. But giving an order is not only a directive, it is also a conditional commissive because if the co-ordination agent delivers in time the customer will pay. This is expressed in axiom D5.

D4. done(dir_a(Customer,AgC,deliver(AgC,

Customer,P,Q,Te),Tb),T)←

done(order(Customer,AgC,P,Q,Te),T).

D5. done(com_cond (Customer,AgC,

deliver(AgC,Customer,P,Q,Te),Te,pay

(Customer,AgC,P,M,Tp),Tp),T)←

done(order(Customer,AgC,P,Q,Te),T).

With axioms A1-A11 and the domain axioms D1-D5 we can describe the communication process between the customer and the co-ordination agent.

John asks that 800 pieces of bearings are processed within maximum 50 days running from the day mentioned in the demand. This demand actually illustrates C1.

C1.
done(product_demand(john,agC,bearing,800
,50),1).

On the second day, the co-ordination agent agC can introduce a fifth day from the demand to meet it, and a delivery period after 40 days. The demand is valid as far as the fifth day. The fact is the following:

C2.
done(product_specification(agC,john,
bearing,800,5,40),2).

From C2 and D3 we can derive that on the second day the co-ordination agent agC authorises John, latest on a fifth day, to request him to deliver, after a 40 -day period , 800 pieces of bearings:

F1.
done(auth(agC,john,deliver(agC,john,
bearing,800,40),5),2)

Now from F1 and A11 we can derive that on a fifth day it holds that from 2 to 5 days John is authorised to request that the co-ordination agent agC delivers 800 pieces of bearings within 40 days:

F2.

holds(Au(john,deliver(agC,john,
bearing,800,40),2,5),5)

John accepts the specification and gives the order in the following fact :

C3.

done(order(john,agC, bearing,800,40),4).

From C3 and D4 then follows:

F3.
done(dir_a(john,agC,deliver(agC,john,
bearing,800,40),40),4)

Now from F3, F2 and A7 we can derive that:

F4.

holds(O(agC,deliver(agC,john,
bearing,800,40),4,40),4)

From C3 and D5 there follows:

F5.

done(com_cond(john,agC,
deliver(agC,john,bearing,800,40),40,pay

(john,agC,bearing,800,45),45),4)←

done(order(john,agC, bearing,800,40),4).

Now from F5 and A10 there follows:

F6.
holds(O(john,pay(john,agC,bearing,800,4)
4,45),4)

The predicates C1 through C3 show the actions that have been performed and the moment they have been performed. From the predicates C1-C3 and the axioms, it can be derived which states of affairs hold at a certain time point, like F2, F4 and F6.

4. Conclusions

This concludes the language for modelling communication and obligations between agents. The language allows the specification of obligations, it supports explicit and dynamic creation of obligations by agents. It does so by means of expressions as directives, commissives, and authorisations. This language has been described in a first order framework.

REFERENCES

1. BENTHEM, J. V. and BERGSTRA, J., **Logic of Transition Systems,** JOURNAL OF LOGIC, LANGUAGE AND INFORMATION, Vol. 3, 1995, pp. 247-283.
2. DIGNUM, F. and WEIGAND, H., **Modelling Communication Between Cooperative Systems**, CaiSE, 1995.
3. GABBAY, **Handbook of Philosophical Logic**, REIDEL, Dordrecht, 1984.
4. LLOYD, J., **Foundations of Logic Programming**, SPRINGER- VERLAG, 1987.
5. MEYER, J.-J. CH. and WIERINGA, R. J., **Applications of Deontic Logic in Computer Science: A Concise Overview** , in J.-J.Ch. Meyer and R. J. Wieringa (Eds.) Deontic Logic in Computer Science: Normative System Specification, WILEY, 1993, pp. 17-40.
6. SEARLE, J. R., **A Taxonomy of Illocutionary Acts, Expression and Meaning: Studies in the Theory of Speech Acts**, CAMBRIDGE UNIVERSITY PRESS, 1979, pp. 1-29.
7. WRIGHT, G. H. v, **Deontic Logic,** MIND, Vol. 60, 1951, pp.1-15.
8. WRIGHT, G. H. v, **A New System of Deontic Logic**, DANISH YEARBOOK OF PHILOSOPHY 1, 1964.