AGENT0: A simple agent language and its interpreter

Yoav Shoham
Computer Science Department
Stanford University
Stanford, CA 94305

Abstract

In [9] we defined the concept of *agent oriented programming* (AOP), which can be viewed as a specialization of *object oriented programming* (OOP). AOP views objects as agents with mental state, and, in the spirit of *speech act theory*, identifies a number of message types – informing, requesting, offering, and so on. AOP is a general framework. In this paper we present a specific and simple language called AGENT0; we define its syntax, present its interpreter, and illustrate both through an example.

Introduction

In [9] I introduce the concept of *agent oriented programming* (AOP). Agents are viewed as computational entities possessing formal versions of of mental state, and in particular formal versions of beliefs, capabilities, commitments, and possibly a few other mentalistic-sounding qualities. A computation consists of these agents informing, requesting, offering, accepting, rejecting, competing with and assisting one another. AOP therefore specializes the *object-oriented programming* (OOP) paradigm, since both frameworks view a computational system as made up of objects with state that pass messages to one another and have individual methods for handling in-coming messages. (I mean OOP in the spirit of Hewitt's original Actors formalism [6], rather than in the more specific sense in which it used today.) Beside renaming objects to be agents, AOP specializes the framework in a number of ways: (a) it fixes the form of the agents' state (now called their mental state), (b) it fixes the form of messages, distinuishing, in the spirit of speech-act theory [2, 8, 4], between informing, requesting, offering, and so on, and (c) it places constraints on methods for responding to incoming messages (agents must be truthful, consistent, etcetera). Figure 1 summarizes the relation between AOP and OOP.

[0]The work was supported by the Air Force Office of Scientific Research and the Stanford Integrated Manufaturing Association.

AOP is a general framework; its most comprehensive instantiation will require dealing with very difficult issues in a variety of areas, including the logic of mental state, belief revision, commitment maintenance, resource managment, and compilation. This paper constitues a modest step by presenting a relatively simple language, called AGENT0. AGENT0, which can be viewed as a base-level agent language, makes explicit the way in which agents are defined and programmed. We also define the interpreter for AGENT0, explicating the flow of control and data structures. AGENT0 embodies simplifying assumptions along several dimensions, including the structure of mental state, the types of communicative commands, and the flow of control in the interpreter.

The structure of this article is as follows. We first briefly review the structure of mental state. In section 3 we provide an overview of agents programs and their interpretation. In sections 4 and 5 we provide more details about the syntax of the language and its interpretation, respectively. In section 6 we show a program fragment illustrating the use of AGENT0. We conclude with a short summary and discussion of related frameworks.

A review of the structure of a mental state

In [10] we discuss the mental state of agents in detail. Here I summarize the few details that directly impact the design of AGENT0. In AGENT0 we consider only two basic mental categories, belief and commitment. The sense of commitment here is that of decision to act, not decision to pursue a goal. We also consider the notion of capability, which strictly speaking is a relation between an agent's mental state and his environment. In AGENT0 we explicitly exclude more complex mental categories such as desires, goals, intentions and plans; this is the first simplifying assumption embodied in AGENT0. The main characteristics of belief, commitment and capability are as follows:

- One both believes a fact *at* a given time and *about* a given time. For example, $\mathtt{B}^{3}_{\mathtt{a}}\mathtt{B}^{10}_{\mathtt{b}}\mathtt{like(a,b)}^{7}$ means that at time 3 agent a believes that at time 10 agent b will believe that at time 7 a liked b.
- Commitments are defined similarly. However, unlike B, CMT has an additional index: $\mathtt{CMT}^{\mathtt{t}}_{\mathtt{a,b}}\varphi$ means that at time t agent a is committed to agent b about action φ. This inter-agent flavor of commitment contrasts with past accounts of the same concept, which viewed it as an intra-agent phenomenon.
- Actions referred to in the context of capability or commitment are always actions at specific times.
- Belief and commitment have internal restrictions; for example, an agent cannot believe contradictory facts nor be committed to incompatible actions.
- Belief and commitment are also mutually constraining. Specifically, an agent is aware of his commitments (i.e., an agent is committed iff he believes himself to be), and an agent only commits in good faith (i.e., if an agent commit to a future fact then he believe the fact will indeed hold).
- An agent is aware of his capabilities (that is, and agent is capable of an action iff he believes himself to be), and only commits to actions of which he is capable.
- Beliefs and commitments, as well as their absence, persist by default. On the other hand, capabilities do not change over time.

It is clear that these properties embody further simplifying assumptions, which may in the future be relaxed.

An overview of AGENT0

The behavior of agents is governed by programs; each agent is controlled by his own, private program. Agent programs are in many respects similar to standard programs, containing primitive operations, control structures and input-output instructions. What makes them unique is that the control structures refer to the mental-state constructs defined previously, and that the IO commands include methods for communicating with other agents.

Framework:	OOP	AOP
Basic unit:	object	agent
Parameters defining state of basic unit:	unconstrained	beliefs, commitments capabilities, ...
Process of computation:	message passing, response methods	message passing, response methods
Message types:	unconstrained	inform, request, offer, ...
Constraints on methods:	none	honesty, consistency, ...

Figure 1: OOP versus AOP

An agent program consists of two parts, *initialization* and *commitment rules*. The initialization defines the capabilities of the agent, its initial beliefs (that is, beliefs at the particular initial time point, but *about* any time) and initial commitments. In other words, this part initializes the mental state (strictly speaking, capability is a relation between mental state and the world, but we ignore this detail).

Commitment rules determine how commitments are added over time. Conditions for making a commitment always refer to the 'current' mental state and the 'current' incoming messages. Actions to which an agent is committed always refer to a particular future point in time.

The order between the commitment rules is completely unimportant, and has nothing to do with the times of action specified in the various rules. This stands in a certain contrast with standard languages. In standard languages there is a simple mapping between the structure of the program and the order of execution; typically, a linear sequence of commands translates to the same execution order. In agent programs, on the other hand, there is complete decoupling between the order among different commitment rules and the time of execution; each commitment rule can refer to an action at any future time. In fact, somewhat paradoxically agent programs never contain direct instructions to execute an action. Rather, agents are continually engaged in two types of activity: making commitments about the future, and honoring previous commitments whose execution time has come (and which have not been revoked in the meanwhile).

In order to implement this process we will make assumptions of two kinds:

1. We assume that the platform is capable of passing message to other agents addressable by name. The program itself will define the form and timing of these messages.
2. Central to the operation of the interpreter is the existence of a clock. The main role of the clock is to initiate iterations of the two-step loop at regular intervals (every 10 milliseconds, every hour, etcetera); the length of these intervals, called the 'time grain,' is determined by the programmer. The other role of the clock is to determine which commitments refer to the current time, and must

therefore be executed.

In AGENT0 we make the very strong assumption that a single iteration through the loop lasts less than the time grain; in the future we will relax this assumption, and correspondingly will complicate the details of the loop itself. Figure 2 presents a flowchart of the AGENT0 interpreter.

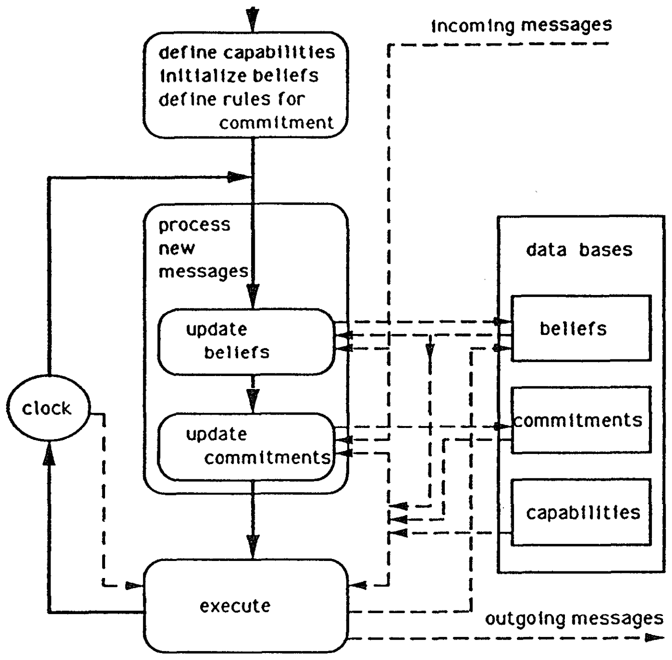

Figure 2: A flow diagram of AGENT0

The syntax of AGENT0

We now take a closer look at the syntax of AGENT0. Full exposition of it is impossible, due to space limitations; that is provided in [9], including a BNF definition. As was said earlier, the first part of the program initializes the mental state of the agent. This is conceptually straightforward and we skip over it here. The bulk of the program consists of the commitment rules. The syntax of a commitment rule is as follows:

```
COMMIT(msgcond,mntlcond,agent,action)
```

where `msgcond` and `mntlcond` are respectively message and mental conditions (see below), `agent` is an agent name, and `action` is an action statement. Mental condition refer to the current mental state of the agent, and message condition refer to messages received in the current computation cycle.

We will discuss the syntax further, but the following simple example of a commitment rule may be helpful at this point:

```
COMMIT(  (?a,REQUEST,?action),
         (B,myfriend(?a)),
         ?a,
         ?action )
```

(Terms preceded by '?' are existentially-quantified variables.) The intuitive reading of the above rule is "if you have just received a request from agent a to take the future action `action`, and you believe a to be friendly, then commit to `action`." The reader might have expected additional conditions for entering into the commitment, such as the requested action being within the agent's capabilities or the absence of contradictory prior commitments. However, as is explained in the next section, these conditions are verified automatically by the interpreter and therefore need not be mentioned explicitly by the programmer.

We do not give the full syntax of mental conditions and message conditions; we only mention that, as is seen in the example, each specify the type (e.g., request, belief) and the content. The example above contains only atomic conditions; in fact, AGENT0 allows for complex conditions which include the logical connective of negation and conjunction.

We conclude the abbreviated description of the syntax with two issues: actions to which an agent may commit, and variables. Regarding action types, we make two orthogonal distinctions: private actions vs. comunicative actions, and conditional actions vs. unconditional ones. Private actions are completely idiosyncratic; examples include rotating a camera platform and retrieving an item from a data base. Communicative actions, on the other hand, are uniform among agents. AGENT0 has only two communicative commands, **INFORM** and **REQUEST**, and an additional **UNREQUEST** type whose effect is to release an agent from an existing commitment. The final unconditional action type is **REFRAIN**, which has no effect on execution, but which blocks commitments to a particular action. Conditional actions are simply actions preceded by a mental condition, or a condition referring to the agent's mental state. The syntax of the mental condition is identical to its syntax in commitment rules, but its interpretation is different in the following respect: in commitment rules mental conditions are evaluated the time at which the commitment is made, whereas in conditional actions they are evaluated at the time of action.

The syntax of action statements is summarized in the following fragment of the BNF definition of the syntax:

```
<action> ::=
DO(<time>,<privateaction>) |
INFORM(<time>,<agent>,<fact>) |
REQUEST(<time>,<agent>,<action>) |
UNREQUEST(<time>,<agent>,<action>) |
REFRAIN <action> |
IF <mntlcond> THEN <action>
```

Finally, as is seen in the earlier example, commitment rules may contain variables. Existentially-quantified variables are denote by the prefix ?. Universally-quantified variables are denoted by the prefix ?!. We do not discuss the roles of the two types of variable further here, but these will be illustrated in the example shown in section 6.

The interpretation of AGENT0 programs

We now discuss the details of interpreter, most of which are quite simple. As in the previous section, we will only be able to mention some of the more important features. In section 4 it was explain that the interpreter operates in cycles, and that each cycle consisted of two phases:

1. Process the incoming messages, updating the beliefs and commitments.
2. Carry out the commitments for the current time.

The second phase is rather straightforward and will not be discussed further here. The first phase is divided into two subphases:

1a. Update the beliefs.

1b. Update the commitments.

To discuss these steps we first need to discuss the representation of beliefs, commitments and capabilities. In AGENT0 they are each represented by a data base. The belief data base is updated either as a result of being informed or as a result of taking a private action; here we discuss only the former update. In AGENT0 agents are completely gullible: they incorporate any fact of which they are informed, retracting previous beliefs if necessary. (This is of course an extreme form of belief revision, and future versions will incorporate a more sophisticated model; see discussion in the final section.) As the result of being informed the agent not only believes the fact, he also believes that the informer believes it, that the informer believes that it (the agent) believes it, and so on. In fact, as the result of informing, the informer and agent achieve so-called *common belief* (the infinite conjunction of "I believe," "I believe that you believe," etcetera). The belief data base will therefore include private beliefs, represented as simple facts, as well as common beliefs, represented by pairs `(a,fact)` (where `a` is the other party).

Items in the data base of capabilities are pairs **`(privateaction,mntlcond)`**. The mental condition part allows one to prevent commitment to incompatible actions, each of which might on its own be possible. An example of an item in the capability data base is

```
([!?time,rotate(?!degree1)],
 NOT (CMT(?!time,rotate(?degree2))
      AND B(NOT ?!degree1=?degree2 )))
```

Items in the data base of commitments are simply pairs `(agent,action)` (the agent to which the commitment was made, and the content of the commitment).

The algorithm for message-induced belief update consists of repeating the following steps for each new incoming `INFORM` message from agent `a` informing of `fact`:

- Add `(a,fact)` to the belief data base;
- If `fact` is inconsistent with the previous beliefs then modify the old beliefs so as to restore consistency.

This last step is of course potentially complicated; both the check for consistency and the restoring of consistency can in general be quite costly, and in general there will be more than one way to restore consistency. We will impose sufficient syntactic restrictions on beliefs so as to avoid these problem. In fact, AGENT0 adopts an extreme restriction, though one that still admits many interesting applications: it disallows in beliefs any connective other than negation. As a result both consistency check and consistency restoration require at most a linear search of the data base, and much less if a clever hashing scheme is used. Other less extreme restrictions are also possible, and will be incorporated in future versions of the language.

Belief change may lead the agent to revoke previous commitments. One reason that might have been expected is that the original commitment relied, among other things, on certain mental conditions stated in the program. These may have included belief conditions that have now changed. Nevertheless, while it would be a natural addition in future versions, in AGENT0 the interpreter is not assigned the responsibility of keeping track of the motivation behind each commitment; that would require a reason-maintenance mechanism that we would rather not incorporate yet. However, while *motivation* is not kept track of, *capability* is. Belief change may remove capabilities, since the capability of each private action depends on mental preconditions. And thus whenever a belief update occurs, the AGENT0 interpreter examines the current commitments to private action, removes those whose preconditions in the capability data base have been violated, and adds a commitment to immediately inform the agents to whom he was committed of this development. Exhaustive examinations of all current commitments upon a belief change can be avoided through intelligent indexing.

It may be surprising to note that the belief update is independent of the program. The update of commitments, on the other hand, depends on the program very strongly, and more specifically on the commitment rules. The algorithm for updating the commitments consists of two steps:

- For all incoming `UNREQUEST` messages, remove the corresponding item from the commitment data base; if no such item exists then do nothing.
- Check all program commitment-statement; for each program statement `COMMIT(msgcond,mntlcond,a,action)`, if :
  - the message conditions `msgcond` hold of the new incoming message,
  - the mental condition `mntlcond` holds of the current mental state,
  - the agent is currently capable of the `action`, and
  - the agent is not committed to `REFRAIN action`, or, if `action` is itself of the form `REFRAIN action'`, the agent is not committed to `action'`.

  then commit to `a` to perform `action`.

An example

The application we choose is a minor modification of one due to John McCarthy [7], who uses it to illustrate his Elephant programming language (see discussion of Elephant in section 6. The example has to do with the communication between a passanger and an airline. The relevant activities of the passanger are querying about flight schedules, making reservations, and collecting boarding passes at the airport. The relevant activities on the part of the airline are supplying information about flight schedules, confirming reservations, and issuing boarding passes. The idea underlying the following program is that confirming a reservation is in fact a commitment to issue a boarding pass if the passanger shows up at the appropriate time.

Since some of the low-level definitions are long, it will be convenient to use abbreviations. We will therefore assume that AGENT0 supports the use of macros. We define the following macros:

```
issue_bp(pass,flightnum,date) ⇒
 IF (B,present(pass)) AND
    B([date/?time],
      flight(?from,?to,flightnum))
 THEN DO(date/?time-1hr,
      physical_issue_bp(pass,flightnum,date))
```

Explanation: This no-frills airline issues boarding passes precisely one hour prior to the flight; there are no seat assignments.

```
query_which(t,asker,askee,q) ⇒
 REQUEST(t,askee,
         IF (B,q) THEN INFORM(t+1,asker,q))
```

Explanation: `query_which` requests only a positive answer; if `q` contains a universally-quantified variable then `query_which` requests to be informed of all instances of the answer to the query `q`.

```
query_whether(t,asker,askee,q) ⇒
 REQUEST(t,askee,
         IF (B,q)
         THEN INFORM(t+1,asker,q))
 REQUEST(t,askee,
         IF (B,NOT q)
         THEN INFORM(t+1,asker,NOT q))
```

Explanation: `query_whether` expects either a confirmation or a discomfirmation of a fact. It is usually a bad idea to include in the fact a universally-quantified variable.

We now define the airline agent. To do so we need to define its initial beliefs, capabilities, and commitment rules.

Of the initial beliefs, the ones relevant here refer to the flight schedule, and the capacity of each flight. The former are represented in the form `[date/time,flight(from,to,number)]` (ignoring the fact that in practice airlines have a more-or-less fixed weekly schedule), and the latter in the form `[date,capacity(flight,number)]`.

The capability relevant here is that issuing boarding passes. Thus the capability data base contains a single item:

```
(physical_issue_bp(?a,?flight,?date),
 (B,[?date,capacity(?flight,?N)]) AND
 (B,?N>|{a:
        ((CMT,?a),
         physical_issue_bp
           (?pass,?flight,?date))}|) )
```

Explanation: `physical_issue_bp` is a private action involving some external events such as printing a boarding pass and presenting it to the passanger. The |...| denotes cardinality.

Finally, the airline agent has two commitment rules:

```
COMMIT(
 (?pass,REQUEST,IF (B,?p)
               THEN INFORM(?t,?pass,?p)),
 (B,?p),
 ?pass,
 IF (B,?p) THEN INFORM(?t,?pass,?p) )
```

```
COMMIT(
 (?pass,REQUEST,issue_bp(?pass,?flight,?date)),
 true,
 ?pass,
 issue_bp(?pass,?flight,?date) )
```

In a more realistic example one would have other commitment rules, notifying the the passenger whether his reservation was confirmed, and the reasons for rejecting it in case it was not accepted. In the current implementation the passenger must query that separately.

This concludes the definition of the simple airline agent. In [9] we trace a simulated execution of this program, starting with several queries and requests of the passenger and culminating with airline issuing a boarding pass at the airport.

Related and future work

There is a large body of work related to the definition of agents and their mental state. However, since that is not the focus of the paper, I do not review that work. To my knowledge there has been less work on programming languages based on a notion of agenthood. One related effort is McCarthy's work on Elephant2000 [7]. This language under development is also based on speech acts, and the airline-reservation scenario I have discussed is due to McCarthy. One issue explored in connection with Elephant2000 is the distinction between illocutionary and perlocutionary specifications, which I have not addressed. In contrast to AOP, in Elephant2000 there is currently no explicit representation of state, mental or otherwise. There is other related work within Distributed AI community (cf. [1]). Although AOP is, to my knowledge, unique in its definition of mental state and the resulting programming language, several researchers have proposed computational frameworks which have to do with commitments (e.g., [11]) and others have made the connection between object-oriented programming and agenthood (e.g., [5, 3]).

We are currently implementing AGENT0 in LISP for the X-windows environment. A Prolog implementation will start soon. At the same time we are engaged in writing experimental programs, in areas as diverse as robotics, traffic control, travel agency planning and construction site management. As a result of these experiments the language will undoubtedly be expanded and modified.

In addition to this exploratory activity we are engaged in several research efforts, which include the following directions:

- Enriching the notion of mental state by additional components such as intention, as well as exploring some of the logical issues that arise in formalizing the various mental components.
- Relaxing various restrictions embodied in AGENT0, such as the restricted form of belief, the naive belief-revision process, and the assumption that the update of mental state lasts less that some fixed time grain.
- Compiling agent programs to a neutral process language.
- Designing agent societies, as opposed to individual agents.

References

[1] Proceedings of the 10th International Workshop on Distributed Artificial Intelligence. Technical Report ACT-AI-355-90, MCC, Austin, Texas, October 1990.

[2] J. L. Austin. *How to Do Things with Words.* Harvard University Press, 1955/1975.

[3] J. Ferber and P. Carle. Actors and agents as reflective concurrent objects: a Mering IV perspective. In [1].

[4] P. Grice. *Studies in the Ways of Words.* Harvard University Press, 1989.

[5] C. Hewitt. Towards open information systems semantics. In [1].

[6] C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8:323–364, 1977.

[7] J. McCarthy. Elephant 2000: A programming language based on speech acts, 1990. unpublished manuscript.

[8] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, Cambridge, 1969.

[9] Y. Shoham. Agent Oriented Programming. Technical Report STAN-CS-90-1335, Computer Science Department, Stanford University, 1990.

[10] Y. Shoham, S. R. Thomas, and A. Schwartz. The mental state of a basic agent, 1990. forthcoming.

[11] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.