Shoham, “Agent-oriented Programming”

§1. Introduction

Present a programming paradigm promoting a social view of computing, where “agents” interact.

§1.1. What is an Agent?

An agent is any entity whose state is viewed as consisting of mental components (e.g., beliefs, capabilities, choices, and commitments).

So agenthood is in the mind of the programmer.

While anything can be viewed as having mental states, it’s not always advantageous to do so.

§1.2. On the Responsible Use of Pseudo-mental Terminology

Elements required to ascribe a given quality to a component of a machine:

- a precise theory regarding the mental category: a semantics that’s clear yet close to the ordinary use of the term;
- a demonstration that the component obeys the theory; and
- a demonstration that the formal theory plays a nontrivial role in analyzing or designing the machine.

The correspondence of the formal theory to common sense needn’t be exact.

§1.3. AOP versus OOP

Use mental constructs to design the computational system.

Mental categories appear in the programming language.

Programming language semantics relates to the semantics of mental constructs.

The agent-oriented programming (AOP) framework specializes the object-oriented programming (OOP) paradigm in the sense of Hewitt's Actors:

view a computational system as composed of communicating modules, each with its own way of handling messages.

AOP fixes the (mental) state of the modules (agents) to consist of components such as beliefs, capabilities, and decisions.

A computation consists of these agents informing, requesting, offering, accepting, rejecting, competing, and assisting one another.

According to speech act theory, each type of communication act involves different presuppositions and has different effects.

Table 1. OOP versus AOP

	OOP	**AOP**
Basic unit	object	agent
Parameters defining state of basic unit	unconstrained	beliefs, commitments, choices, …
Process of computation	message passing and response methods	message passing and response methods
Types of messages	unconstrained	inform, request, offer, promise, decline, …
Constraints on methods	none	honesty, consistency, …

§2. Two Scenarios

The first scenario is complex -- the type of application envisioned.

The second is a toy example serving three purposes:

- It crisply illustrates several AOP ideas.
- It's implementable in the simple AGENT-0 language defined later.
- It illustrates the fact that agents needn't be robotic agents.

§2.1. Manufacturing Automation

Agents:

- Alfred handles regular-order cars.
- Brenda handles special-order cars.
- Calvin is a welding robot.
- Dashiel is a coordinating program controlling the plant.

- 8:00: Alfred requests that Calvin promise to weld ten bodies for him that day.

  Calvin agrees to do so.

- 8:30: Alfred requests that Calvin accept the first body, Calvin agrees, and the first body arrives.

  Calvin starts welding it and promises Alfred to notify him when it is ready for the next body.

- 8:45: Brenda requests that Calvin work on a special-order car which is needed urgently.

  Calvin responds that it cannot right then. but that it will when it finishes the current job, at approximately 9:00.

- 9:05: Calvin completes welding Alfred's first car, ships it out, and offers to weld Brenda's car.

  Brenda ships it the car, and Calvin starts welding.

- 9:15: Alfred enquires why Calvin is not yet ready for his (Alfred's) next car.

  Calvin explains why, and also that it (Calvin) expects to be ready by about 10:00.

- 9:55: Calvin completes welding Brenda's car, and ships if out.

  Brenda requests that it reaccept it and do some painting, but Calvin refuses, explaining that it does not-know how to paint.

  Calvin then offers to weld another car for Alfred, and proceeds to weld Alfred's cars for a while.

- 12:15: Brenda requests that Calvin commit to welding four more special-order cars that day.

  Calvin replies that it cannot, since that conflicts with its commitment to Alfred, who still has six unwelded cars.

  Brenda requests Alfred to release Calvin from its commitment to Alfred. Alfred refuses.

  Brenda requests that Dashiel (remember Dashiel?) order Calvin.to accept her important request and revoke its commitment to Alfred.

  Dashiel orders Calvin to weld two of Brenda's cars, and then as many of Alfred's as time allows.

- 3:30: Calvin senses it is low on lubricant, and, without interrupting its work, asks Alfred for when the next lubrication is scheduled.

  Alfred responds that he doesn't know, but that Dashiel should know.

  Calvin asks Dashiel, who informs it that the next lubrication is scheduled for 4:00.

  Since that is sufficiently soon, Calvin continues to weld.

----

§2.2. Airline Reservation

Agents:

- P is a passenger.
- C is an airline clerk, a program.
- S is C's supervisor.

By confirming a reservation, the airline enters into a commitment to issue a boarding pass to the passenger at the appropriate time.

<u>March</u>

- P to C: Please inform me what flights you have from San Francisco to New York on April 18.
- C to P: Flight #354 departs at 08:30. flight #293 departs at 10:00, flight #441 departs at noon, …
- P to C: Please book me on #354.
- C to P: That is sold out.
- P to C: Please book me on #293.
- C to P: That is confirmed: your reservation number is 112358.
- P to C: Please book me also on #441.
- C to P: That conflicts with #293: I am not allowed to double book a passenger.
- P to C: Please get permission to do so.
- C to S: I request permission for the fallowing double booking: …
- S to C: Permission denied.
- C to P: Sorry, I cannot get approval.

<u>April 18, at the airport</u>

- P to C: My name is P; I have a reservation for flight =293.
- C to P: Here is your boarding pass.

§3. Overview of the AOP Framework

A complete AOP system includes three primary components:

1. a restricted formal language (including several modalities such as belief and commitment) with clear syntax and semantics for describing mental states;
2. an interpreted programming language in which to define and program agents, with primitive commands (e.g., REQUEST, INFORM);
3. an "agentifier", converting neutral devices into programmable agents.

Component 2 is the main purpose of this article.

It relies on component 1.

Component 3 remains rather mysterious.

§4. Mental Categories and Their Properties

There's no "correct" selection of mental categories nor a "correct" theory of them.

§4.1. Components of Mental State

The actions of an agent are determined by its *decisions*, or *choices*.

Decisions are constrained (but nor determined) by the agent's *beliefs*, which refer to

- states of the world,
- mental states of other agents, and
- *capabilities* of this and other agents.

Decisions are also constrained by prior decisions.

So we introduce two mental categories, *belief* and *decision*, and a third (not *per se* mental) category, *capabilities*.

Rather than take decision as basic, we start with *obligation*, or *commitment*, and treat decision as commitment to oneself.

§4.2. A Language for Belief, Obligation, and Capability

Time

We believe things both *about* different times and *at* different times.

Likewise for other modalities.

We adopt a point-based temporal language – e.g.,

```
holding(robot,cup)^t
```

means

"The robot is holding the cup at time *t*."

Action

We don't distinguish between actions and facts:

the occurrence of an action is represented by the corresponding fact holding.

E.g., instead of saying that the robot took the action raise-arm at time *t*, we say that the sentence

```
raise-arm(robot)^t
```

is true.

(To retain the agency behind the action, we introduce the notion of decision.)

Since actions are facts, they're instantaneous.

Belief

Using modal operator *B*,

$B_a^t j \quad \Rightarrow$ "At time *t* agent *a* believes that φ",

where φ is a (recursively defined) sentence.

E.g.,

$$B_a^3 B_b^{10} like(a,b)^7$$

means

"At time 3 agent *a* believes that at time 10 agent *b* will believe that at time 7 *a* liked *b*."

Obligation

$$OBL_{a,b}^t j$$

means

"At time *t* agent *a* is obligated (committed) to agent *b* about φ."

Decision (choice)

Decision is defined to be obligation to oneself:

$$DEC_a^t j \quad =_{def} \quad OBL_{a,a}^{\,t} j$$

Capability

$CAN_{a}^{t} j$

means

“At time *t* agent *a* is capable of φ.”

E.g.,

$CAN_{robot}^{5} open(door)^{8}$

means

“At time 5 the robot can ensure that the door is open at time 8.”

`ABLE` is the “immediate” version of `CAN`.

Letting `time(φ)` be the outermost time occurring in sentence φ,

$$ABLE_{a} j \quad =_{def} \quad CAN_{a}^{time(j)} j$$

§4.3. Properties of the Various Components

Internal Consistency

We assume that both the beliefs and the obligations are internally consistent:

- For any *t*, *a*: $\left\{ j : B_a^t j \right\}$ is consistent.
- For any *t*, *a*: $\left\{ j : OBL_{a,b}^t j \text{ for some } b \right\}$ is consistent.

Good Faith

Agents commit only to what they believe themselves capable of and only if they mean it:

- For any *t*, *a*, *b*, φ: $OBL_{a,b}^t j \quad \rightarrow \quad B_a^t((ABLE_a j) \wedge j)$

Introspection

Agents are aware of their obligations:

- For any t, a, b, φ: $OBL_{a,b}^t j \Leftrightarrow B_a^t OBL_{a,b}^t j$.
- For any t, a, b, φ: $\neg OBL_{a,b}^t j \Leftrightarrow B_a^t \neg OBL_{a,b}^t j$.

But we don't assume that agents need be aware of commitments made *to* them.

Persistence of Mental State

Consider how mental states persist or change.

- **Beliefs** persist by default: agents have perfect memory of their beliefs; a belief is dropped only when a contradictory fact is learned.

  The absence of belief also persists by default.

- **Obligations** persist but default, but there are conditions under which they're revoked, e.g.:
  - explicit release of the agent by the party to which it's obligated, and
  - realization by the agent that it's no longer able to fulfill the obligation.

- Since **decision** is defined in terms of obligation, it inherits the default persistence.

  While an agent can't unilaterally revoke obligation to others, it can cancel obligations to it -- including decisions.

- We here assume that **capabilities** are fixed.

The Contextual Nature of Modal Statements

Skip.

§4.4. A Short Detour: Comparison with Cohen and Levesque

Skip.

§5. A Generic Agent Interpreter

The role of an agent program is to control the evolution of an agent's mental state.

Actions occur as side-effects of the agent being committed to an action whose time has come.

The Basic Loop

Each agent iterates the following steps at regular intervals:

1. Read the current messages and update your mental state, including your beliefs and commitments.

(The agent program is crucial for the update.)

2. Execute the commitments for the current time, possibly resulting in further belief change.

(This is independent of the agent program.)

Actions to which agents can be committed include

- communicative actions (e.g., informing and requesting) and
- arbitrary "private" actions.

See **Figure 1**.

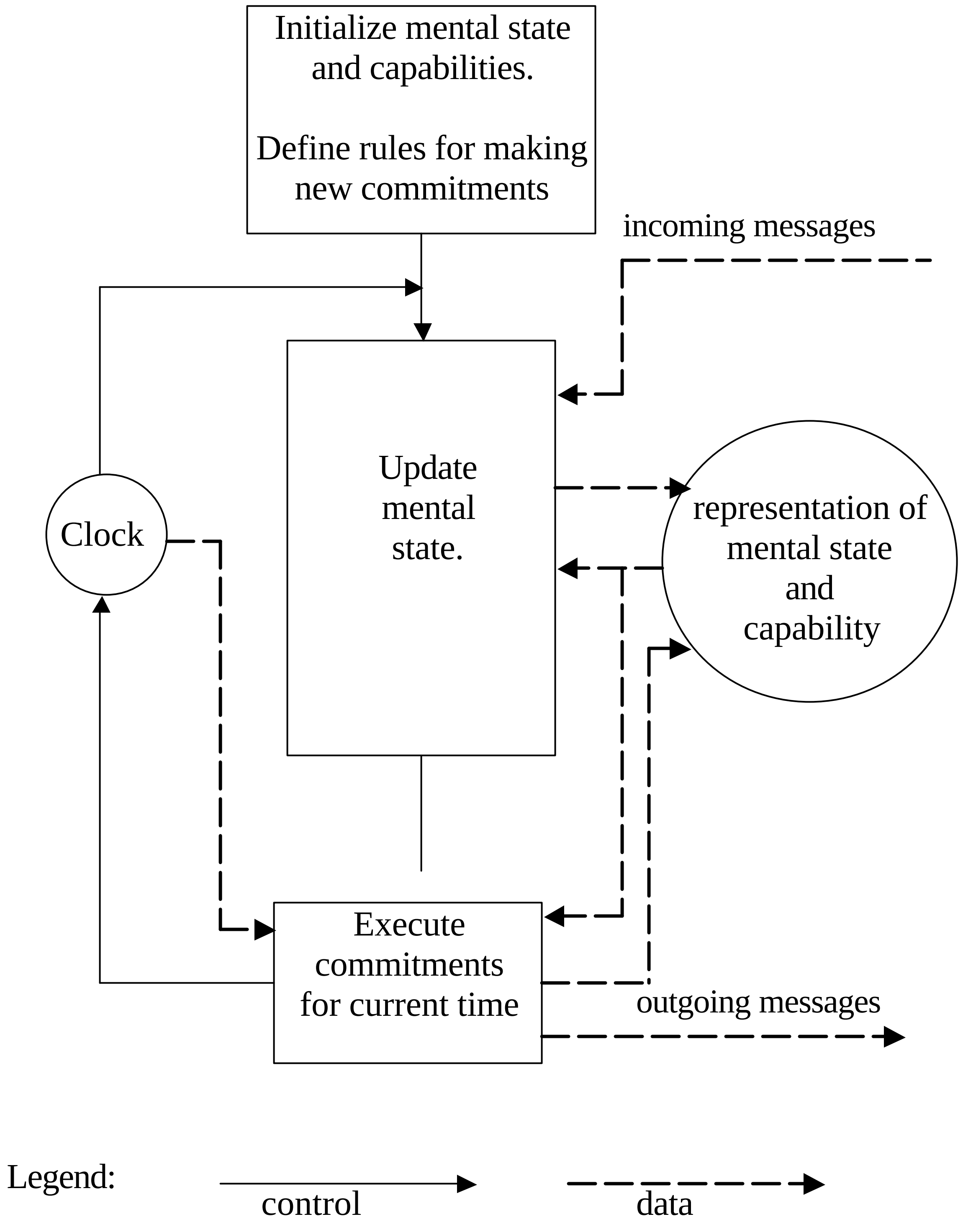
Initialize mental state and capabilities.
Define rules for making new commitments
incoming messages
Update mental state.
Clock
representation of mental state and capability
Execute commitments for current time
outgoing messages
Legend:
control
data

Fig. 1. A flow diagram of a generic agent interpreter

Assumption about Message Passing

Assume that the platform can pass messages to other agents. addressable by name.

The interpreter determines when messages are sent.

Assumption about the Clock

The clock initiates iterations of the two-step loop at regular intervals.

The length of these intervals (the “time grain”) is determined by the settable variable `timegrain`.

Assume a variable `now` whose value is set by the clock to the current time.

Assume that a single iteration through the loop lasts less than the time grain.

(A very strong assumption)

Synchronization is crucial for proper functioning of a society of agents.

§6. AGENT-0, A Simple Language and Its Implementation

§6.1. The Syntax of AGENT-0

The language itself specifies only conditions for making commitments.

Commitments are actually made, and later carried out, automatically at the appropriate times.

Commitments are only to primitive actions (directly executed by the agent).

So an agent can't commit to achieving any condition that requires planning.

Fact Statements

Fact statements are used to specify both the contents of actions and conditions for their execution.

They're the atomic objective sentences of the temporal language described above:

```
(t (employee smith acme))
(NOT (t (employee jones acme)))
```

Private and Communicative Action Statements

Actions may be *private* or *communicative* and, independently, *conditional* or *unconditional*.

The syntax for a private action (e.g., by a database agent or a robot) is

```
(DO  t  p-action)
```

where `t` is a time point and `p-action` is a private action name.

Private actions may or may not involve IO.

Communicative actions always involve IO and are common to all agents.

AGENT-0 has only three types of communicative action:

- The syntax of **informing** is

  ```
  (INFORM  t  a  fact)
  ```

  where `t` is a time point, `a` an agent name, and `fact` a fact statement.

- The syntax of **requesting** is

  ```
  (REQUEST  t  a  action)
  ```

  where `action` is an action statement, recursively defined.

  E.g.,

  ```
  (REQUEST  1  a  (DO  10  update-database))
  (REQUEST 1  a  (REQUEST  5  b
                       (INFORM  10  a  fact)))
  ```

- The syntax of **canceling** a request is

  ```
  (UNREQUEST  t  a  action)
  ```

We can also prevent commitment to a particular action:

```
(REFRAIN  action)
```

Conditional Action Statements

We distinguish between

- commitments for conditional actions, which include conditions to be tested just before acting, and
- conditions for entering into commitments in the first place (see below).

A conditional action relies on a *mental condition*, which refers to the mental state of the agent.

When the time comes to execute the action, the mental state at that time is examined to see whether the mental condition is satisfied.

So the agent and time components of the mental state are left implicit.

A mental condition, then, is any combination of modal statements in the temporal-modal language, with the primary *agent* and *time* arguments omitted.

Specifically, a mental condition is a logical combination of *mental patterns* of the form

`(B fact)` or `((CMT a) action)`

("`CMT`" means the same as "`OBL`".)

E.g.: `(B (t (employee smith acme)))`

The syntax of a conditional action, then, is

```
(IF  mntlcond  action)
```

E.g.,

```
(IF  (B  (t′  (employee  smith  acme)))
     (INFORM  t  a
             (t′  (employee  smith  acme))))
```

Mental conditions may contain the logical connectives `AND`, `OR`, `NOT`.

E.g., the following three actions constitute a query about whether `fact` is true (`b` is being queried and is asked to inform `a`):

```
(REQUEST  t  b  (IF (B fact)
                    (INFORM  t+1  a  fact)))

(REQUEST  t  b  (IF (B (NOT fact))
         (INFORM t+1 a (NOT fact))))

(REQUEST  t  b
         (IF (NOT (BW fact))
             (INFORM  t+1  a
                 (NOT (t+1 (BW  a  fact))))))
```

Variables

Procedures are invoked in a pattern-directed fashion.

Commitment rules are activated by certain patterns (generally involving variables) in the incoming messages and current mental state. (See below.)

A variable begins with "`?`".

It may range over agent names, fact statements, or action statements.

E.g.,

```
(IF (NOT ((CMT ?x) (REFRAIN sing))) sing)
```

Variables in action statements are interpreted as existentially quantified.

The scope of the quantifier is upwards to the scope of the first `NOT` or (if the variable isn't in the scope of a `NOT`) it's the entire statement – see the last example.

A universally quantified variable begins with "`?!`"; its scope is the entire formula.

E.g.,

```
(IF (B (t (emp ?!x acme)))
     (INFORM t′ a (t (emp ?!x acme))))
```

Commitment Rules

Most of the action statements are unknown at programming time – they’re communicated by other agents.

The program itself just contains conditions for the agent to enter into new commitments.

Most commitments are in response to messages.

Conditions for commitments include both mental conditions (see above) and message conditions (referring to the current incoming messages).

A message condition is a *message patterns* of the form

```
(From  Type  Content)
```

where

- `From` is the sender’s name,
- `Type` is `INFORM`, `REQUEST`, or `UNREQUEST`, and
- `Content` is a fact statement or an action statement, depending on `Type`.

Other information associated with a message (destination and arrival time) is left implicit.

E.g:

```
(a  INFORM  fact)
```

means that one of the new messages is from `a` informing the agent of `fact`.

And

```
(AND  (a  REQUEST  (DO  t  walk))
    (NOT (?x  REQUEST  (DO  t  chew-gum))))
```

means that there’s a message from `a` requesting the agent to walk and but no new request from anyone that the agent chew gum.

The syntax of a commitment rule is

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

(The action statement may contain its own mental condition.)

E.g.,

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

A program is a sequence of commitment rules, preceded by a definition of the agent's capabilities and initial beliefs, and the fixing of the time grain.

The following is the BNF for the AGENT-0 syntax.

```
<program>  ::=
     timegrain := <time>
     CAPABILITIES := (<action> <mntlcond>)*
     INITIAL BELIEFS := <fact>*
     COMMITMENT RULES := <commitrule>*
<commitrule>  ::=
     (COMMIT <msgcond> <mntlcond>
             (<agent> <action>)*)
<msgcond>  ::=
    <msgconj>  |  (OR <msgconj>*)
<msgconj>  ::=
    <msgpattern>  |  (AND <msgpattern>*)
<msgpattern>  ::=
    (<agent> INFORM <fact>)    |
    (<agent> REQUEST <action>  |
    (NOT <msgpattern>)
<mntlcond>  ::=
    <mntlconj>  |  (OR <mntlconj>*)
```

```
<mntlconj>  ::=
    <mntlpattern>  |  (AND <mntlpattern>*)
<mntlpattern>  ::=
    (B <fact>)                |
    ((CMT <agent>) <action>)  |
    (NOT <mntlpattem>)
<action>  ::=
    (DO        <time> <privateaction>)   |
    (INFORM    <time> <agent> <fact>)    |
    (REQUEST   <time> <agent> <action>)  |
    (UNREQUEST <time> <agent> <action>)  |
    (REFRAIN   <action>)                 |
    (IF        <mntlcond> <action>)
<fact>  ::=
    (<time> (<predicate> <arg>*))
<time>  ::=
    <integer>  |  now  |  <time-constant>    |
    (+ <time> <time>)  |  (- <time> <time>)  |
    (× <iD.teger> <time>)
      ; Time may be a <variable> when
      ; it appears in a commitment rule
<time-constant>  ::=
    m  |  h  |  d  |  y
      ; m (minute) =60, h (hour) 3600, etc.
<agent>  ::=
    <alphanumeric_string>  |  <variable>
<predicate>  ::=  <alphanumeric_string>
<arg>  : :=
    <alphanumeric_string>  |  <variable>
<variable>  ::=
    ?<alphanumeric_string>   |
    ?!<alphanumeric_string>
```

§6.2. The AGENT-0 Interpreter

Since the AGENT-0 interpreter is an instance of the generic interpreter, it inherits the two-step loop design.

In AGENT-0, the mental state consists of three components.

One – capabilities – is fixed.

So the first step in the loop may be specialized as:

(1a) Update the beliefs.

(2a) Update the commitments.

In AGENT-0, the beliefs, commitments, and capabilities of an agent are each represented by a database.

Updating Beliefs

The belief database is updated either

a. as a result of being informed or

b. as a result of taking a private action -- e.g.,

1) A database agent comes to believe a fact after performing a retrieval operation.
2) A robotic agent comes to believe something after performing a visual routine.

We're interested in **a**.

Consider assimilating a new fact φ into an existing database Γ.

Checking consistency for unconstrained theories (databases) is either intractable (in the propositional case) or undecidable (in the first-order case).

If φ is inconsistent with Γ, most theories of assimilation require that Γ be "minimally" modified to restore consistency – this is an even harder problem.

There are at least two approaches to getting around the complexity:

1. Relax the requirements. Adopt a heuristic assimilation algorithm that compromises soundness or completeness.

2. [Taken in AGENT-0] Restrict the sentences so the problem becomes tractable.

AGENT-0 disallows connectives other than negation – consistency checking is at most linear in the database size.

(This is in addition to disallowing modal operators, needed for nested beliefs.)

There remains the question of how to judge the new information – we ultimately need a theory of authority.

AGENT-0 agents incorporate any fact they're told, retracting the contradictory atomic belief if it was held.

Updating Commitments

Items in the database of commitments are pairs

`(agent action)`,

the agent to which the commitment was made and the content of the commitment.

Items in the database of capabilities are pairs

`(privateaction mntlcond)`.

The mental condition part prevents commitment to incompatible actions.

E.g.,

```
((?!time (rotate wheelbase ?degrees))
 (NOT ((CMT ?x) ?!time (service wheelbase))))
```

Existing commitments are removed either

a. as a result of `UNREQUEST` messages

The agent removes the corresponding item from the commitment database if it exists, else does nothing.

or

b. as a result of belief change.

Belief change may affect capabilities since the capability of each private action depends on mental preconditions.

So, whenever a belief update occurs, the AGENT-0 interpreter examines the current commitments to private actions.

It removes those whose preconditions in the capability database have been violated.

It should add a commitment to inform the agent to which it was committed (but AGENT-0 doesn't enforce this).

Removing existing commitments is independent of the program, but adding them depends on the program.

Algorithm to add commitments

For each program commitment statement

`(COMMIT msgcond mntlcond (a`$_i$` action`$_i$`)*)`

if

- `msgcond` holds of the new incoming messages,
- `mntlcond` holds of the current mental state,
- for all `i`, the agent is currently capable of `action`$_i$, and
- for all `i`,
  - the agent is not committed to `REFRAIN action`$_i$, and
  - if `action`$_i$ is itself of the form `REFRAIN action`$_i$, the agent isn't committed to `action`$_i$

then, for all `i`, commit to `a`$_i$ to perform `action`$_i$.

Conditions for an agent to be capable of an action:

- An agent can request and unrequest anything from anyone.
- An agent can inform anyone of a fact he believes.

  An agent can inform *itself* of any fact at all (useful to implement reasoning in the agent).
- An agent is capable of any private action in the capability database provided the mental condition associated with that private action by the database holds at that time.
- An agent can refrain from any action provided he's not already committed to it.
- An agent can perform a conditional action `(IF mntlcond action)` if he can perform `action` under the condition `mntlcond`.

Carrying Out Commitments

Each commitment in the commitment database has an associated time.

In this second step, the interpreter executes all actions whose time is in the interval (`now-timegrain`, `now`].

The meaning of "execute" depends on the type of action:

- `INFORM`, `REQUEST`, `UNREQUEST`: Send the appropriate message.
- `REFRAIN`: No effect on execution.
- `DO`: Consulting the belief and commitment databases, check the mental condition associated with the primitive action in the capability database; if it holds, then perform the primitive action.
- `IF`: Consulting the belief and commitment databases, test the mental condition; if it holds, then (recursively) execute the action.

§6.3. A Sample Program and Its Interpretation

Present a program implementing a simplified version of the airline representative of §2.

The ideas behind the program are that

- the relevant activity on the part of the airline is issuing a boarding pass to the passenger, and
- confirming a reservation is a commitment to issue a boarding pass at the appropriate time.

We first define some macros.

```
------------------------------------------------------------------------------
(issue_bp pass flightnum time) ⇒
  (IF (AND (B ((- time h) (present pass)))
           (B (time
                (flight ?from ?to flightnum))))
      (DO time - h
          (physical_issue_bp
              pass flightnum time)))
```

Explanation: Issue the boarding pass precisely 1 hr. (`h`) before the flight.

`physical_issue_bp` is a private action involving some external events.

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

Explanation: This 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)
          (INFORM (+ t 1) asker q)))
   (REQUEST t askee
      (IF (B (NOT q))
          (INFORM (+ t 1) asker (NOT q))))
```

Explanation: `query_whether` expects either a confirmation or a disconfirmation of a fact.

Now, to define the airline agent, we define its initial beliefs, capabilities, and commitment rules.

Initial Beliefs

Concerning the flight schedule:

```
(time (flight from to number))
```

And the number of seats available:

```
  (time (remaining_seats time1 flight_number seats)
```

Capabilities

These are issuing boarding passes and updating the count of the available seats on flights.

So the capability database contains two items:

```
 ((issue_bp ?a ?flight ?time) true)

 ((DO ?time
      (update_remaining_seats ?time1 ?flight_number
                              ?additional_seats))
  (AND
     (B (?time
           (remaining_seats ?time1
                  ?flight_number ?current_seats)))
     (?current_seats >= |?additional_seats|)))

   ; update_remaining_seats is a private action
   ; changing the belief about remaining_seats.
```

Commitment Rules

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

(COMMIT
   (?cust REQUEST
      (issue_bp ?pass ?flight ?time))
   (AND
      (B (?time (remaining_seats ?flight ?n)
      (?n > 0)
      (NOT ((CMT ?anyone)
           (issue_bp ?pass ?anyflight ?time))))
   (myself
      (DO (+ now 1)
         (update_remaining_seats
            ?time ?flight -1)))
   (?cust (issue_bp ?pass ?flight ?time)))
```

See **Table 2**: a sample exchange between a passenger and the airline agent.

The messages from the passenger are determined by him.

Those of the airline are initiated by the agent interpreter in response.

Table 2
Sample exchange between a passenger and an airline agent

agent	action
smith	(query_which lmarch/l:00 smith airline (18april/?!time (flight sf ny ?!num)))
airline	(INFORM lmarch/2:00 smith (18april/8:30 (flight sf ny #354)))
airline	(INFORM lmarch/2:00 smith (18april/10:00 (flight sf ny #293)))
airline	(INFORM lmarch/2:00 smith (18april/ …
smith	(REQUEST lmarch/3:00 airline (issue_bp smith #354 18april/8:30))
smith	(query_whether lmarch/4:00 smith airline ((CMT smith) (issue-bp smith #354 18april/8:30)))
airline	(INFORM lmarch/5:00 smith (NOT ((CMT smith) (issue-bp smith #354 18april/8:30))))
smith	(REQUEST lmarch/6:00 airline (issue-bp smith #293 18april/10,.00))
smith	(query-whether lmarch/7:00 smith airline ((CMT smith) (issue-bp smith #293 18april/10:00)))
airline	(INFORM lmarch/8:00 smith ((CMT smith) (issue-bp smith #293 18april/10:00)))
…	
smith	(INFORM 18april/9:00 airline (present smith))
airline	(DO 18april/9:00 (issue-bp smith #293 18april/10:00))

§6.4. Implementation

Skip.

§7. Agentification

Releasing manufacturers from the requirement to supply a mental state creates a gap between

- the intentional level of agent programs and
- the mechanistic process representation of a given device.

The role of the agentifier is to bridge this gap.

We follow Rosenschein and Kaebling – *situated automata* – in this decoupling of the intentional and machine levels.

There's

- a low-level language for describing the device and
- a high-level language for the designer to reason about the device.

The compiler takes a program in the high-level language and produces a description of a device in the low-level language.

For the low-level process language, we require

- representation of process time, including real-valued durations,
- asynchronous processes, and
- multiple levels of abstraction.

– He has developed his own process model, *temporal automata* (see references).

In agentification,

- the input to the translator includes a description of a machine in the process language and
- the output is an intentional program.

So compilation into situated automata is *de*-agentification.

§8. Related Work

Skip.

§9. Discussion

Directions in which to extend the framework:

· Mental categories

Augment the language of mental states to include more complex notions (e.g., desires, intentions, plans).

This will allow

- ◊ a richer set of communicative commands and
- ◊ more structure on the behavior of agents.

· Groundedness of mental categories

One contribution of distributed computing to the formal theory of knowledge is the concrete grounding of the semantics:

the set of possible worlds became the set of possible global states of a collection of finite-state processes, given a protocol.

With agentification, we should be able similarly to anchor belief and commitment.

· Probability and utility

Most work on knowledge and belief adopts crisp notions of mental attitudes – no representation of graded belief or commitment.

This contrasts with game-theoretic work on rational interaction among agents in economics and AI, where uncertainty and utility play key roles.

· Inheritance and groups

An analogue to inheritance in OOP would be (in AOP) "group agents," a group of agents constituting an agent.

If we define

- ◊ the beliefs of the composite agent as the common beliefs of the individual agents and
- ◊ the commitments of the composite agent as the common commitments of the individual agents,

then mental attitudes of the group are indeed inherited by the individuals.

· Persistence of mental states

Dealing formally with the persistence of mental states is even harder than dealing with the frame problem (cf. §4).

E.g., if I believe that you don't believe *x*, do I believe that you won't believe *x* in a little while?

· Resource limitations

The definition of the interpreter assumed that belief and commitment updates take negligible time.

But in many real-time applications this assumption is violated.

So the interpreter should choose wisely among mental operations.

(There's much interest in intelligent real-time problem solving, including the tradeoff between quality and timeliness.)

· Belief revision and update

AGENT-0 accepts all new information.

But we should consider what constitutes a reasonable policy of belief update.

There are both semantic and algorithmic questions.

· Temporal belief maps

AGENT-0 can't represent beliefs of agents about the beliefs or commitments of other agents.

AGENT-0 keeps track of its beliefs by a *time map,* recording the points of transition and assuming default persistence between them.

AGENT-1 allows nested modalities in the belief database, using high-dimensional time maps called *mental time maps.*

Temporal belief maps are a special case.

· Societies

We've looked only at agents functioning autonomously.

But successful agent societies need some global constraints – e.g.,

- ◊ social *rules* and
- ◊ social *roles* .

Both of these reduce the problem solving required by agents and the communication overhead.

There's a rich body of literature on computer societies – e.g.,

- ◊ Minsky's informal Society of Mind metaphor
- ◊ Winograd's studies of societal roles (human and machine)
- ◊ Moses and Tennenholz's discussion of the computational advantages of social laws
- ◊ Doyle's work on the relationship between AI, rational psychology, and economics

Shoham *et al*. have recently investigated the off-line design of social laws that strike a good balance between

- ◊ preventing chaos and
- ◊ allowing sufficient freedom to individual agents.

They're currently investigating the automatic on-line learning of such laws.