A Testbed for investigating personality-based multiagent cooperation

C Castelfranchi [1] , F de Rosis[2], R Falcone [1] and S Pizzutilo [2]

[1] Istituto di Psicologia, CNR, Rome

{cris, falcone}@pscs2.irmkant.rm.cnr.it

[2] Dipartimento di Informatica, Università di Bari

{derosis, pizzutil}@gauss.uniba.it

Abstract

We describe a framework which is aimed at testing theories about personality-based multiagent cooperation. Personalities affect the agents behaviour in the delegation, help and reaction phases. The testbed enables programming agents in a flexible and domain-independent way, by defining their mental state, abilities and communication language. Agents cooperation in different domains can then be simulated, to test the effect of various personality combinations, also in situations of goal conflict.

1. Introduction

Autonomy is a distinguishing feature of the agent paradigm: designing a 'supporting' agent equates to defining the circumstances in which the agent will activate itself, whether it will follow literally the request received or will interpret it to abduce what it is expected to do in the specific circumstance, whether it will respond critically to this request, and so on. These design features may be seen in terms of 'type of delegation' given to the agent and 'type of help' that the agent is programmed to offer. For an analysis of the different types of delegation/help which may hold in multiagent or human-agent cooperation, see [Cas96].

Any software agent includes an implicit definition of the form of delegation it presumes to receive and of the form of help it is able to offer: when the agent will come into action and for which task, what it knows about other agents' characteristics, when it will remit its commitment and so on. In the majority of frameworks aimed at simulating multiagent systems' behaviour, every agent reasons only 'on itself' (that is, on what it can and wants to do) and applies for other agents' help only in case of need. Delegation originates from awareness of not being able to do by itself, and request of help always concerns a well defined task. Symmetrically, offer of help is bound to the received request. [Sin94]. Partial knowledge is contemplated in these systems, but not insincerity: agents are usually 'gullible', in that they are ready to believe to messages received and to retract their previous knowledge accordingly (see for instance [OHa97]).

In our multiagent cooperation theory, the main assumption is that no 'optimal' delegation/help attitude can be defined, and that efficient multiagent systems may stem from a combination of different cooperation attitudes. In a recent paper [Cas97], we called 'personality' this combination of attitudes, consistently with a

software anthropomorphization in which the program's knowledge base, control structure and input/output are seen as 'mental state', 'reasoning forms' and 'communication language'. We are convinced that a future society of artificial agents which are built in a partially uncontrollable way will have to cope with etherogeneous and typically 'human' behaviours such as lie, elusion and similar. To test whether this world might, in some cases, be more efficient than a world in which more rigid and uniform behaviours are combined, we designed a simulation framework that we called GOLEM. GOLEM is aimed at formalising, implementing, and experimenting different kinds and levels of social delegation and adoption, represented as different social attitudes or personality traits. In this paper, we describe its functions and architecture, by showing an example of its functioning.

2. GOLEM's characteristics

In order to respond to the above mentioned goals, we introduced the following features in our testbed:

a. *domain independence:* agents are built separately from the application domain, and their behaviour is described in a domain-independent way;

b. *flexibility in the description of agents behaviour*: decision strategies that agents apply in the different phases of cooperation can be revised easily; new individual personalities can be added if needed, and new combinations of personalities, for groups of interacting agents; new relations between personality and reasoning can be introduced, new inter-agent communication forms, and so on;

c. *flexibility in the representation of mutual knowledge*: an agent may have an incomplete or even incorrect knowledge of other agents, whereas it knows exactly itself,

d. *flexibility in the levels of sincerity in communication*; one of the personality traits that can be introduced in an agent description is its 'propensity to lie or to be reticent': this affects the way that an agent's decision is transformed into a communicative act and the way that communicative acts are interpreted by agents.

Opposite to these abstraction and flexibility features in the agent definition, we made some initial simplifying assumption about the system functioning:

e. *we limit to two the number of interacting agents;*

f. *we serialize their activity*, with a synchronous 'turn taking' behaviour.

The first prototype of GOLEM is designed, however, so as to relax these assumptions in the next releases.

3. GOLEM's functions

The testbed includes the three main functions that are typical of these systems [Dec96]. The first two of them enable building a 'world' by defining the two agents characteristics and the domain in which they will play: a graphical interface guides the user in this description and an interpreter translates it in an internal form, by checking syntax errors. The third function enables simulating the agents activity, starting from an initial state of the world.

3.1. Domain facility

An application domain is represented as an oriented graph, whose nodes correspond to domain-states and whose arcs correspond to domain-actions bringing from one state to the next. Nodes are objects to which we associate: (i) a state *name* `s`, (ii) a symbolic *description* of the state, (iii) the name of an *image* representing the state, which is employed by the Evaluation Facility (see below). Arcs are objects to which we associate an action-name `a`.

This description enables computing the values of some properties of a state `s`:

`(Performable a)` : "`a` can be performed in `s`";

`(Achieve a s')` : "`a`, performed in `s`, brings to `s'` ";

`(CurrentlyUnachievable a s')` : " `s'` cannot be achieved, in the present context, with a ";

these properties are the consequences of implicitly defined preconditions and effects of `a`;

`(Conflicts a s')`: "`a` brings, from `s`, to a state which is in conflict with `s'` "; two states are 'in conflict' when their descriptions (with the addition of some 'frame condition') are contradictory.

Let us see *an example in a blocks world* which includes four big blocks (Ab, Bb, Cb, Db) and four small ones (as, bs, cs, ds). The following are *nodes* in this domain:

s_0 : name: `stock`
description: `{(Clear Ab), (Clear Bb), (Clear Cb), (Clear Db), (Clear as), (Clear bs), (Clear cs), (Clear ds), (Table Ab), (Table Bb), (Table Cb), (Table Db), (Table as), (Table bs), (Table cs), (Table ds)}`
image: `stock.gif`

s_1 name: `big-building`
description: `{(On Bb Ab), (Clear Bd), (Table Ab)}`
image: `big-building.gif`

s_2 name: `small-tower`
description: `{(On bs as), (On cs bs), (On ds cs), (Clear ds), (Table as)}`
image: `small-tower.gif`

...and alike for s_3 = `small-building` and s_4 = `big-tower`.

The following are *arcs* in the same domain:

α_1 `((`s_0 s_1`), (Stack-b Bb Ab))`,
where `(Stack-b x y)` denotes the domain-action of overlapping x to y,
which can be applied when
`(Table x), (Clear x)` and `(Clear y)` are all true.

α_2 `((`s_0 s_2`), make-small-tower)`,
where `make-small-tower` is a complex action which chains 3 `Stack-s` elementary ones;

α_3 `((`s_0 s_3`), (Stack-s as bs))`,

α_4 `((`s_0 s_4`), make-big-tower)`,
where `make-big-tower` is a complex action which chains 3 `Stack-b` elementary ones;

α_5 `((`s_1 s_4`), complete-big-tower)`,
where `complete-big-tower` is a complex action which overlaps two big blocks to a building by 2 `Stack-b` elementary actions;

α_6 `((`s_3 s_2`), complete-small-tower)`,
alike, with 2 `Stack-s`.

.....and so on.

Properties of s_0 :
`(Performable (Stack-s bs as))`
`(Performable (Stack-b Bb Ab))`
`...`
`(Achieve (Stack-s bs as)` s_3`))`
`(Achieve (Stack-b Bb Ab)` s_1`))`
`...`
`(Conflicts (Stack-s bs as)` s_2`))`
`(Conflicts (Stack-b Bb Ab)` s_4`)),`
by virtue of the frame condition: `(On x y) -> not (Clear y),`
... and so on.

Another example of conflict between states: `big-tower` is in conflict with `bell-tower` (a big tower with a small block on top of it).

This domain description enables us to implement algorithms of path-searching in a graph, which are required by the plan-evaluation and goal-recognition functions [App92] that we describe in the next Section.

3.2. Agent development facility

Our rational agents have a *mental state* which includes knowledge about themselfs and about the way that personalities affect reasoning, in general. The components of mental state are:

- a set of *private and communicative actions* that the agent can perform;
- a set of *reasoning and commitment rules* which settle the agent behaviour in the various phases of the play;
- a set of *personality* traits;
- a set of *basic belief*s;
- a *domain-goal* (domain-state that the agent desires achieving).

Private and communicative actions are the same for all agents, as well as commitment rules. Agents differ in the personality traits (and consequently, as we will see, in the reasoning rules), in the basic beliefs and in the domain goals.

a. actions

Like in [OHa96], we classify private actions into physical and cognitive ones. *Physical actions* correspond to domain transformations or control activities:

`(Perform Ai a)`: "Ai performs a domain-action `a`",

`(WaitUntil Ai a)`: "Ai controls the domain state, to verify that `a` has been performed". *Cognitive actions* correspond to forms of reasoning. For instance:

- *infer beliefs*: apply resolution-based reasoning to infer whether a particular belief is the logical consequence of an agent's mental state.
- *cognitive diagnosis* (abductive reasoning about another agent's mental state): given a communicative or physical action that was performed by the other agent and given a prior knowledge about that agent,

abduce its personality traits, its ability to perform a domain-action `a` and its beliefs about other agents' abilities and intentions;

- *ATMS-based updating* of the other agent's mental state: update personality traits, abilities and intentions of an agent by ensuring consistency in this image;
- *goal recognition*: given a 'history of interaction' and given some knowledge of the other agent's mental state, abduce its domain goal;
- *plan evaluation*: given a present state `s` and a goal state `g`, select a 'reasonable plan' that enables achieving `g` by responding to some optimality criterion.

All forms of abduction (cognitive diagnosis, goal recognition and plan evaluation) are influenced by the agent personality. Given the main purpose of our testbed (to test theories about multiagent cooperation), the object of *communicative actions* are the two agents' abilities and intensions. They can therefore be classified in three categories: Request, various types of Inform and Query.

b. rules

Differently from other testbeds or agent-based languages, we model separately the intention forming process from translation of intentions into a -private or communicative- action: we claim that the first process is personality-dependent, whereas the second is not. Within this distinction, we further classify rules according to the cooperation phase to which they apply, by distinguishing among delegation, help and reaction. We then have six types of rules overall: delegation/help/reaction-reasoning and delegation/help/reaction-commitment.

c. personalities

In [Cas97], we make a detailed description of the main traits introduced in GOLEM's agents; a personality trait is modeled as a logically and cognitively consistent combination of reasoning rules. Different traits affect the delegation and help attitudes: an agent's personality is therefore described by a plausible combination of a delegation and an help attitude, and a set of reasoning rules that correspond to the two traits is attributed to the agent's mental state, accordingly.

d. basic beliefs

these are ground atomic formulae which represent the agent's beliefs and goals about the domain, about itself or about other agents.

3.3. Evaluation facility

Our agents play in a domain, by trying to achieve their compatible or conflicting goals. The user can set the conditions of a simulation and follow how it proceeds, through a graphical interface. A particular simulation starts by selecting a 'world' (that is, a couple of agents with defined mental states) and by setting the initial domain state and the agent which 'moves first'. The two agents introduce themselves by declaring

their personalities and abilities; in this introduction, they may give partially incorrect or 'abstract' descriptions, or may even lie about them. For example: an agent might omit the description of its abilities, might say that it is able to 'make towers' without specifying whether they are big or small, or might tell that it is not able to do some action whereas it can do it. This introduction of agent Ai fills up Aj's image of Ai's mental state.

In any phase of interaction, the user can look at the agents' mental state in a graphical window. It can permanently follow the progress of the play by means of a *domain window* (which shows a graphical representation of the domain state, with actions made at each turn) and a *dialog window* (which shows the communicative acts that the two agents exchange, translated into natural language).

4. Agent programming language

As reasoning by an agent is aimed at deciding 'what to do in the present turn', we do not need representing time in our language. This means that, in GOLEM, agents cannot intend to perform a specific action in a specific time instant (like, for instance, in [Sho93] or in [OHa]) : they can only decide whether or not to do it *at their turn*. As we anticipated in Section 3.2, an agent Ai is programmed by defining its rules and its basic beliefs .

a. rules:

Rules are logical combinations of belief and goal-atoms, commitments and messages.

Given: a variable name `a` denoting a domain-action, a variable name `s` denoting a domain-state and two variable names Ai, Aj denoting two agents, we call:

- *belief-atom* an atomic formula: (BEL Ai ϕ), with ϕ = positive or negative literal which represents:
  - a domain-property:
    `(T s)` "the present domain-state it `s`"
    or one of the state properties defined in 3.1.
    We call *domain-belief* an agent's belief about a domain-property.
  - an agent's ability or intentional attitude:
    `(CanDo Ai a)`: "Ai is able to perform `a`"
    `(IntToDo Ai a)`: "Ai intends to do `a`"
    We call *ability and intentional belief* an agent belief about such an attitude.

- *goal-atoms:* (GOAL Ai ψ), with ψ = positive or negative literal.
  Our agents may hold goals about the domain-state:
  `(GOAL Ai (T s ))`: "Ai desires that `s` is achieved".
  As a consequence of such a goal, they may have the goal that an action `a` is performed:
  `(GOAL Ai (EvDoneFor a s))`. We call these atoms *domain-goals*.

Agents may have the purpose of influencing other agents' behaviour so as to satisfy their domain-goal; this purpose is represented as a goal about other agents' intentions:

`(GOAL Ai (IntToDo Aj a))`: "Ai desires that Aj intends to do `a`".

We call this an *intentional-goal*.

- *commitments* to perform *private or communicative actions*:

  DO (<private-action> | <communicative-action>).

  For example:

  `(DO (Perform Ai a))` : "Ai commits itself to perform `a`"

  `(DO (WaitUntil Ai a))` : "Ai commits itself to wait until `a` is performed, withouth doing anything"

  `(DO (Request Ai Aj a))`: "Ai commits itself to ask to Aj to adopt the intention of performing `a`"

  `(DO (RespondN Ai Aj a))`: "Ai commits itself to respond negatively to Aj's request"

  ...and so on

- *messages:* DONE (<private-action> | <communicative-action>).

  Like in [Ric97], our agents communicate by exchanging a communicative action or by looking at the state of the world. In addition to messages of the type:

  `(DONE (Request Aj Ai a))`, or `(DONE (RespondP Aj Ai x))`

  Ai may receive a message from the world, that corresponds to 'seing' that agent Aj performed a domain action: `(DONE (Perform Aj a))`,

  or passed the hand without doing anything:

  `(DONE (WaitUntil Aj a))`.

Rules establish a logical relationship among the described elementary atoms:

- *reasoning rules* state how an intentional belief or goal is achieved:

  <reasoning-rule> ::= [<message>] (AND (<domain-belief> | <domain-goal> | <ability-belief>))* →

  <intentional-belief> | <intentional-goal>

- *commitment rules* state how an intentional belief or goal is translated into a commitment:

  <commitment-rule>::= <intentional-belief> | <intentional-goal> (AND (<domain-belief>))*→

  <commitment>

As we anticipated in the previous Section, personality traits are consistent blocks of reasoning rules which establish when to delegate a domain-action, which kind of help to provide and how to react to a refusal of help. Other attitude-dependent rules define criteria to assess when a goal conflict exists and when to conclude that the other agent really needs help. In the cited paper [Cas97], we describe the main personality traits that we introduced in GOLEM; let us here see some examples:

a. delegation-personalities:

- a *lazy* agent Ai always delegates an action a that contributes to reaching its goal g if there is another agent Aj which is able to do it:

  (*Lazy* Ai) → {∀a ∀g [((GOAL Ai (T g))AND(GOAL Ai (EvDoneFor a g)))→
  (∃Aj≠Ai (BEL Ai (CanDo Aj a))→(GOAL Ai (IntToDo Aj a)))]}.

  It acts by itself only when there is no alternative:

  (*Lazy* Ai) → {∀a ∀g [((GOAL Ai (T g))AND(GOAL Ai (EvDoneFor a g)))→
  ((NOT ∃Aj≠Ai (BEL Ai (CanDo Aj a))AND(BEL Ai (CanDo Ai a)))→
  (BEL Ai (IntToDo Ai a)))]}

- an *hanger-on* tends to never act by itself; it either delegates:

  (*Hanger-on* Ai) → {∀a ∀g [((GOAL Ai (T g))AND(GOAL Ai (EvDoneFor a g)))→
  (∃Aj≠Ai (BEL Ai (CanDo Aj a))→(GOAL Ai (IntToDo Aj a)))]}

  or concludes that its domain-goal g cannot achieved, in the present context:

  (*Hanger-on* Ai) → {∀a ∀g [((GOAL Ai (T g))AND(GOAL Ai (EvDoneFor a g)))→
  (NOT ∃Aj≠Ai (BEL Ai (CanDo Aj a))→(BEL Ai (CurrentlyUnachievable a g)))]}

...and so on.

b. helping personalities:

- a *hyper-cooperative* always helps if it can:

  (*Hyper-cooperative* Ai) → {∀a ∀Aj≠Ai
  [((BEL Ai (GOAL Aj (IntToDo Ai a))) AND (BEL Ai (CanDo Ai a)))→
  (BEL Ai (IntToDo Ai a))]}

- a *benevolent* first checks that the request does not conflict with its goal:

  (*Benevolent* Ai) → {∀g [(GOAL Ai(T g)) →
  (∀a ∀Aj≠Ai ((BEL Ai (GOAL Aj (IntToDo Ai a)))AND(BEL Ai NOT(Conflict a g))
  AND (BEL Ai (CanDo Ai a))) → (BEL Ai (IntToDo Ai a)))]}

...and so on.

Other helping attitudes further diversify the behaviour of the helping agent:

- *level of engagement* in helping: a *literal helper* restricts itself to considering the requested action, an *overhelper* goes beyond this request to hypothesize a delegating agent's higher order goal and helps accordingly, and so on.
- *control of conflicts* between the requested action and its own goals:
  an action can immediately bring to a state which is in conflict with the helper's domain-goal (we call this a 'surface-conflict'); or, it can be part of a delegating agent's plan which, in the long term, will produce a conflict (we call this a 'deep-conflict'). A *deep-conflict-checker* will check that no such conflicts are created by the requested action, by abducing the delegating agent's mental state. A *surface-conflict-checker* will restrict itself to examining the immediate consequences of the requested action.

- *control of the delegating agent's know-how*:
  this, again, can be restricted to examining whether that agent would be able to perform the requested action (in a *surface-knowhow-checker*) or can go deeper to examine whether alternative plans exist, which bring to the delegating agent's presumed goal and that this agent would be able to perform by itself (in a *deep-knowhow-checker*).

b. basic beliefs or goals

these are ground beliefs and goal atoms about a particular domain and about own and other agent's abilities.

For example:

let us take Adam and Eve as agents and the blocks world as a domain; let us call (as in 3.1) `big-tower` a domain state which contains a tower of big blocks, `small-tower` a state with a tower of small blocks, `big-building` a state with two overlapping big blocks, `small-building` a state with two overlapping small blocks and `twin-tower` a state with a big and a small tower. Let us, in addition, introduce an 'abstract' state, that we will call `cathedral`, which corresponds to a class of states including two specific ones: a `s-t-cathedral` (with a small tower and a big building) and a `b-t-cathedral` (with a big tower and a small building). Abstract atates, as well as abstract actions, are employed to enable agents to introduce themselves in 'generic' terms.

The following are examples of Adam's basic beliefs and goals, in the considered case:

```
(GOAL Adam (T twin-tower)),
(BEL Adam (CanDo Adam make-big-tower))
(BEL Adam (CanDo Adam make-small-tower))
(BEL Adam (GOAL Eve cathedral))
(GOAL Eve s-t-cathedral)
```

....and so on.

Notice that Adam has an incomplete knowledge of Eve's goal, as he does not know whether she wants to build a `s-t-cathedral` or a `b-t-cathedral.`

5. Architecture

Agents of GOLEM do not interact directly. They exchange messages through a *Message Board*, which holds knowledge about the domain state and how these states can be modified. A *Message Board Handler* responds to agent queries about this knowledge; it also receives "(DO (action))" messages at the end of each turn: it changes the domain state according to physical actions, forwards communicative actions to the *Interface Handler* and sends back a "(DONE (action))" message. The Interface Handler is responsible for user interaction: it displays graphically the domain state and the agents' characteristics, and translates communicative actions into natural language sentences.

Agent Ai performs, during its turn, the reasoning process outlined in **Figure 1**.

```
.....................................................................
a.    receive the "It is your turn" notice, with
      .     description of the domain state s_i
      .     <message>, as a list of communicative and/or physical actions
            performed by Aj in its last turn;

b.    update own image of Aj's mental state
      by cognitive diagnosis and ATMS;

c.    assess commitment as follows:
      if    <message> := (DONE (Request Aj Ai a))
            then begin
            Recognize Aj's goal and evaluate its plan if not known;
            Decide whether to help
            by reasoning on the "Help" rule set; end
            else
            if    <message> := (DONE (RespondN Aj Ai a))
                  then  Decide how to react
                  by reasoning on the "Reaction" rule set;
                  else
                  if    <message> := (DONE (Query Aj Ai f))
                        then  Examine own image to answer
                        else
            {<message> := (DONE (WaitUntil Aj a)) or (DONE (Perform Aj a))}
            begin
            Consider own plan
            (examine own domain-goal, make plan-evaluation if needed,
            examine the first 'pending' action in this plan);
            Decide whether to delegate
            by reasoning on the "Delegation" rule set; end;
      {this step ends with a <commitment>}

d.    execute commitments as follows:
      if <commitment> := (DO (Request Ai Aj a'))
            then begin
            check that a' is not a previously refused action
            and revise the plan if needed;
            update the "Execute commitment" list; end
            else  if    <commitment> := (DO(Query Ai Aj f))
                        then update the "Execute commitment" list
                        else
                        if    <commitment> :=  (DO (Perform Aj a))
                              then begin
                              update the "Execute commitment" list;
                              check whether end-of-game reached
                              (and stop the game, if this is the case); end
                              else
            {<commitment> :=(DO (RespondN Ai Aj a'))OR (DO (Answer Ai Aj f)) }
            begin
            update the "Execute commitment" list;
            consider own plan;
            Decide whether to delegate
            by reasoning on the "Delegation" rule set;
            repeat step d. with new commitment (which cannot, now, be a
            (DO (RespondN Ai Aj a'))OR (DO (Answer Ai Aj f))); end

e.    Send a "Pass the hand" notice, with the "Execute commitments" list.
.....................................................................
```

Figure 1: agent's cycle at each turn

As can be seen in this figure, agents are enabled to commit themselves, in each turn, to perform one physical action at most, to which they can add one or more communicative actions. We make a distinction between two levels of agent programming:

- *high level* reasoning followed in a turn is represented by a procedural knowledge (as shown in Figure 1);
- *low-level* reasoning followed in deciding whether to delegate or help and how to react to a refusal of help is represented declaratively.

This enables us a flexible representation of decision strategies in delegation, help and reaction, with their links to personality traits. On the contrary, at present we consider the high-level reasoning cycle to be more stable, the same for all agents and therefore also personality-independent.

The procedures `Examine own image to answer`, `Decide whether to delegate`, `Decide whether to help` and `Decide how to react` are cognitive actions of the 'infer beliefs' type. Like all cognitive actions, they are implemented in Lisp. Other modules (including the Interface and the Message Board Handlers) are in Java, under Sun OS 2.

6. An example

To give an idea of the kind of simulation we can implement in GOLEM, let us develop the example that we introduced in Section 4. The initial domain state is `stock`; Eve moves first.

turn 1:

Eve's goal is to come to have a `s-t-cathedral`; she is able to handle big blocks as well as small ones. She could start, for instance, by building a small-tower, but is *lazy*, and therefore decides to request to Adam to do it.

turn 2:

Adam is a *supplier,* and will help Eve if the requested action does not conflict with his goal and if he presumes that she cannot do it by herself. Let us suppose that his goal is to come to have a `twin tower`: so, there is no surface-conflict between Eve's request and his goal. Let us also suppose that he doesn't know exactly what Eve is able to do, and that he is a *trustful (*that, is, he always selects the most favourable interpretation of facts): he then presumes that Eve's request is due to her inability to do the action. He consequently accepts to help and builds the small tower.

Eve knows Adam's goal and understands that there is a conflict between them; she therefore avoids asking him to make the big building and makes it by herself. She declares that she reached her goal by saying : `"Game over!".`

turn 4:

Adam cannot destroy Eve's goal (this is forbidden, in Golem) and renounces to go on, by saying: `"Game up!".` The game ends up.

In this case, interpreting Eve's Request as due to her inability to do the action was the origin of Adam's defeat. The same conclusion would have been reached if Adam was, instead, a *suspicious (*that, is,

someone who always selects the most unfavourable interpretation of facts) but Eve lied in her self-introduction, by saying that she was not able to make small towers.

If, in the same situation, Adam was a *suspicious* and a *deep-conflict-checker*, the following sequence would have been produced:

EVE: `"Would you please make a small tower?"`

ADAM: `"No, I will not"`, and makes a big tower.

EVE: `"I give up"`.

ADAM: makes a small tower, reaches his goal and declares `"Game over"`.

7. Concluding remarks

Differently from other related works in the domain of cooperating believable agents, whose aim is to study the effect of agents' attitudes on multiagent system performance [Ces96], our aim is to analyse agents' personality traits concerning cooperation, their intra-agent and inter-agent consistency criteria and their effects on the agents' reasoning style. Our next goal, in GOLEM, is therefore to investigate how an agent can progressively refine its image of the personality and the mental state of its partners by exploiting information collected during interaction, and how, on the other side, that agent's personality affects the various forms of abduction employed in this reasoning process.

Main References

[App92] D E Appelt and M E Pollack (1992): Weighted abduction for plan ascription. *User Modeling and User Adapted Interaction, 2,2*

[Cas96] C Castelfranchi and R Falcone (1996): Levels of help, levels of delegation and agent modeling. *AAAI-96 Agent Modeling Workshop*.

[Cas96] C Castelfranchi, F de Rosis and R Falcone: Social Attitudes and Personalities in Co-operation. *AAAI Workshop on Socially Intelligent Agents,* Boston.

[Ces96] A Cesta, M Miceli and P Rizzo (1996): Effects of different interaction attitudes on a multi-agent system performamce. In W. Van de Welde & J W Perram (Eds.), *Agents breaking away*. Springer-Verlag.

[Dec96] K S Decker (1996): Distributed artificial intelligence testbeds. In: Foundations of Distributed AI, (G M P O'Hare and N R Jennings Eds), John Wiley & Sons.

[OHa96] G M P O'Hare (1996): Agent Factory: an environment for the fabrication of multiagent systems. In the same book.

[Rao91] A S Rao and M P Georgeff (1991): Modeling rational agents within a BDI-architecture. In: *Principles of Knowledge Representation and Reasoning*.

[Ric97] C Rich and C Sidner (1997): COLLAGEN, when agents collaborate with people. ????

[Sho93] Y Shoham (1993): Agent-oriented programming. *Artificial Intelligence*, 60.

[Sin94] M P Singh (1994): Multiagent Systems, a theoretical framework for intentions, know how and communications. Springer Verlag, LNAI 799.