PRACTIONIST: a Framework for Developing BDI Agent Systems

Vito Morreale*, Susanna Bonura*, Giuseppe Francaviglia*,
Michele Puccio*, Fabio Centineo*, Giuseppe Cammarata*,
Massimo Cossentino†, and Salvatore Gaglio†‡

*R&D Laboratory - ENGINEERING Ingegneria Informatica S.p.A.
†ICAR-Italian National Research Council
‡DINFO-University of Palermo

I. The framework

In this abstract we give a brief overview of the PRACTIONIST framework, which supports programmers in developing BDI agents and is built on top of JADE [1], a widespread platform that implements the FIPA[1] specifications. Therefore, our agents are deployed within JADE containers and their main cycle is implemented by means of a JADE cyclic behaviour (figure 2).

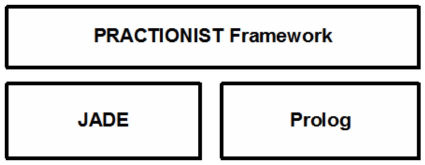


Fig. 1. PRACTIONIST over JADE and Prolog.

A PRACTIONIST agent is a software component endowed with the following elements (figure 2):

- a set of *perceptions* and the corresponding *perceptors* that listen to some relevant external stimuli;
- a set of *beliefs* representing the information the agent has got about both its internal state and the external environment;
- a set of *goals* the agent wishes or wants to pursue. They represent some states of affairs to bring about or activities to perform and will be related to either its desires or intentions (see below);
- a set of *goal relations* the agent uses during the deliberation process and means-ends reasoning;
- a set of *plans* that are the means to achieve its intentions;
- a set of *actions* the agent can perform to act over its environment; and
- a set of *effectors* that actually execute the actions.

As shown in figure 2, PRACTIONIST agents are structured in two main layers: the framework defines the execution logic and provides the built-in components according to such a logic, while the top layer includes the specific agent components to be implemented, in order to satisfy system requirements.

[1]http://www.fipa.org

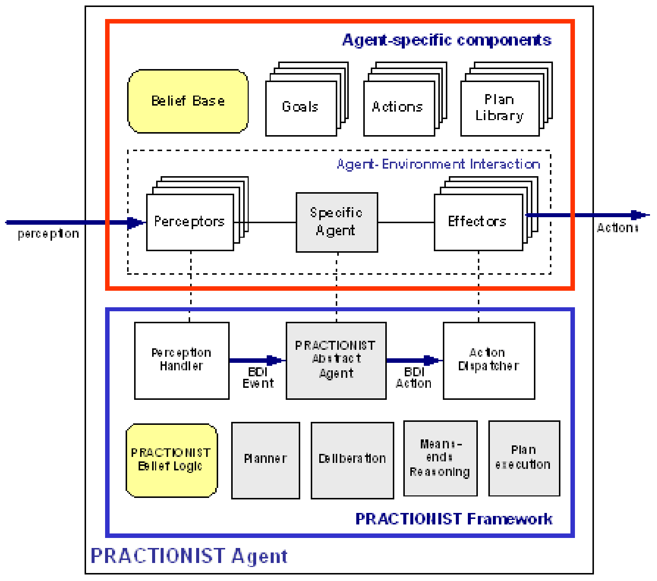


Fig. 2. Components of PRACTIONIST agents.

Therefore, a developer who wants to design an agent has to develop *(i)* the *Goals* the agent could pursue, *(ii)* the means (a set of plans, i.e. the *Plan Library*) to pursue such goals or to react to the stimuli coming from the environment, *(iii)* the *Perceptors* to receive such stimuli, *(iv)* the *Actions* the agent could perform and the corresponding *Effectors*, and *(v)* the set of beliefs and rules (*Belief Base*) to model the information about both its internal state and the external world (details on beliefs are given in [2]).

In the following section we give an overview of how to program some of agent components, with reference to the paper ”*Reasoning about Goals in BDI Agents: the PRACTIONIST Framework*”, presented at the WOA 2006 [3].

II. Implementing agent components

The concepts and the examples given in this section refer to the *tileworld* demonstrator, which is a multi agent system with two types of agents, i.e. an agent that manages the environment and player agents.

Several simulation parameters can be altered at run time, such as the appear rate and the life cycle of holes, tiles and obstacles. These information was represented by beliefs [2] about the state of the environment represented through the following predicates:

- *gridSize(width: X, height: Y)* represents the size of the grid, in terms of width and height,
- *holeBirth(rate: X)* and *holeLifecycle(rate: X)* represent the frequency of holes' birth and their mean life cycle,
- *tileBirth(rate: X)* and *tileLifecycle(rate: X)* represent the frequency of tiles' birth and their mean life cycle,
- *obstacleBirth(rate: X)* and *obstacleLifecycle(rate: X)* represent the frequency of obstacles' birth and their mean life cycle,
- *agent(name: X)* represents other active player agents.

The framework provides the support to let agent make meta-level reasoning. In other words, each player agent, by reasoning on above information, will be able to select the optimal strategy to increase its score. For example, the plan *FindTileInAmplitudePlan* implements a depth search behavior, while the plan *FindTileRandomicallyPlan* implements a random search strategy. Thus these plans are used by the player to find a tile in several circumstances.

Analogously, agent beliefs about its state refer to the following predicates:

- *position(xPos: X, yPos:Y)* represents the position of the player agent,
- *score(value: X)* represents the current score of the player,
- *hold(obj: tile)* states that the player agent holds a tile.

On the base of such beliefs, some goals are defined as well. As an example, the *HoldTile* is a state goal that succeeds when *hold(obj: tile)* is believed true by the agent for the same *tile*. Thus, in the *AchieveTilePlan*, the player agent has to identify a tile within the grid to satisfy the *HoldTile* goal and then hold such a tile by executing the action of picking it up.

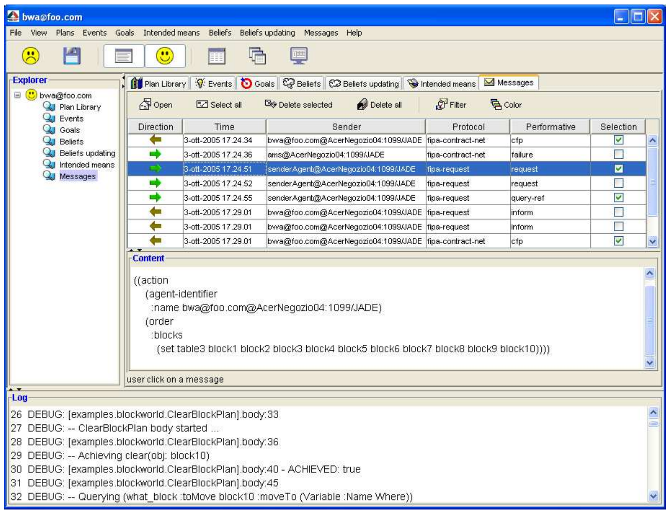
The player agent is endowed with the *Taker* effector, which triggers and executes the pick up action and updates the environment status and its internal state. The agent is also provided with other effectors (e.g. *Mover*, *Releaser*, etc.) to be able to perform other actions, such as moving itself in the grid and releasing holding tiles.

Finally, the cognitive system of the agent includes a set of perceptors that receive stimuli from the environment. As an example, the player agent is equipped with the perceptors *TileLifeCyclePerceptor*, *HoldBirthPerceptor*, etc. to be able to perceive changes from the environment about tiles' birth rate, the obstacles' life cycle, and so forth.

III. PRACTIONIST Agent Introspection Tool (PAIT)

The framework also provides developers with the PRACTIONIST Agent Introspection Tool (PAIT), a visual integrated

[2]In PRACTIONIST beliefs can be about either predicates or other beliefs (expressed by the operator Bel). Moreover, predicates can be expressed by specifying the role of their arguments, i.e. $predicate(role1 : element1, role2 : element2, ..., roleN : elementN)$.



Direction	Time	Sender	Protocol	Performative	Selection
←	3-ott-2005 17.24.34	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-contract-net	cfp	✓
→	3-ott-2005 17.24.36	ams@AcerNegozio04:1099/JADE	fipa-contract-net	failure	☐
→	3-ott-2005 17.24.51	senderAgent@AcerNegozio04:1099/JADE	fipa-request	request	✓
→	3-ott-2005 17.24.52	senderAgent@AcerNegozio04:1099/JADE	fipa-request	request	☐
→	3-ott-2005 17.24.55	senderAgent@AcerNegozio04:1099/JADE	fipa-request	query-ref	✓
←	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-request	inform	☐
←	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-request	inform	☐
←	3-ott-2005 17.29.01	bwa@foo.com@AcerNegozio04:1099/JADE	fipa-contract-net	cfp	✓

```
((action
   (agent-identifier
     :name bwa@foo.com@AcerNegozio04:1099/JADE)
   (order
     :blocks
       (set table3 block1 block2 block3 block4 block5 block6 block7 block8 block9 block10))))
```

user click on a message

```
26  DEBUG: [examples.blockworld.ClearBlockPlan].body:33
27  DEBUG: -- ClearBlockPlan body started ...
28  DEBUG: [examples.blockworld.ClearBlockPlan].body:36
29  DEBUG: -- Achieving clear(obj: block10)
30  DEBUG: [examples.blockworld.ClearBlockPlan].body:40 - ACHIEVED: true
31  DEBUG: [examples.blockworld.ClearBlockPlan].body:45
32  DEBUG: -- Querying (what_block :toMove block10 :moveTo (Variable :Name Where))
```

Fig. 3. The PRACTIONIST Agent Introspection Tool (PAIT).

monitoring and debugging tool, which supports the analysis of the agent's state during its execution. In particular, the PAIT can be suitable to display, test and debug the agents' relevant entities and execution flow. Each of these components can be observed at run-time through a set of specific tabs (see figure 3); the content of each tab can be also displayed in an independent window.

All the information showed at run-time could be saved in a file, providing the programmer with the opportunity of performing an off-line analysis. Moreover, the PAIT provides an area for log messages inserted in the agent source code, according to the Log4j approach. The usage of this console and the advantages it provides are described in more details in [4].

Acknowledgments

This work is partially supported by the Italian Ministry of Education, University and Research (MIUR) through the project PASAF.

References

[1] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE - a FIPA-compliant agent framework," in *Proceedings of the Practical Applications of Intelligent Agents*, 1999. [Online]. Available: http://jmvidal.cse.sc.edu/library/jade.pdf

[2] V. Morreale, S. Bonura, G. Francaviglia, M. Cossentino, and S. Gaglio, "PRACTIONIST: a new framework for BDI agents," in *Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS'05)*, 2005, p. 236.

[3] V. Morreale, S. Bonura, G. Francaviglia, F. Centineo, M. Cossentino, and S. Gaglio, "Reasoning about goals in BDI agents: the PRACTIONIST framework," in *Proceedings of Joint Workshop "From Objects to Agents"*, 2006.

[4] V. Morreale, S. Bonura, F. Centineo, A. Rossi, M. Cossentino, and S. Gaglio, "PRACTIONIST: implementing PRACTIcal reasONIng sySTems," in *Proceedings of Joint Workshop "From Objects to Agents"*, 2005.