Multi-Agent System Architecture with Planning for a Mobile Robot.

Bianca Innocenti[1], Beatriz Lopez[1], and Joaquim Salvi[2]*

[1] Agents Research Lab.
http://eia.udg.es/arlab
[2] Computer Vision and Robotics
http://iiia.udg.es/VICOROB.html
Institut d'Informàtica i Aplicacions, Universitat de Girona,
Avda Lluís Santaló S/N E-17006 Girona, Spain.
{bianca,blopez, qsalvi}@eia.udg.es

Abstract. This paper focuses on the development of a multi-agent system in which several planning agents provide alternative plans for a given situation. This multi-agent system constitutes, at a higher abstraction level what is called the Task Planning Agent, an agent that belongs to a Multi-Agent Architecture used to control one single mobile robot.

1 Introduction

The challenge of developing autonomous robots involves several related problems, such as dynamic modelling of the world, task and path planning, planning and scheduling, etc.

Traditionally, each problem has been solved and implemented in a module based architecture, where the relationships among all components are established at the design stage [10] [17]. This kind of architecture constrains in some way the possible outcome when the robot has to perform a task. This had led researchers to either focus on new, more flexible architectures, or to develop collections of autonomous robots that coordinate their activities in order to solve complex tasks. In the latter case, multi-agent architectures have been applied in order to produce global behavior in an agent population. An agent in such an approach is equivalent to a robot [15] [14].

In our current research, we proposed a robot architecture based on a multiagent system (MAS). Here, all the agents make up single robot. The agents have the same global goal: to control the robot and to do it intelligently, while competing for resources. We believe that our approach will produce a more robust, flexible, reusable, generic and reliable architecture that can be easily modified and completed to permit social behavior among robots; it is also in line with the research in holonic multi-agent systems [9] [13].

* This work has been partially supported by the Spanish MCyT under project TIC2001-4936-E.

The paper is organized as follows: in Section 2 we explain the MAS architecture used to control the robot; in Section 3 we give details of its implementation; in Section 4 the MAS planner system is illustrated using an example; and finally, in Section 5 we give some conclusions and discuss future work.

2 Multi-Agent Architecture

We propose a Multi-agent Architecture to control a single robot composed of certain specific agents. These are the task planning agent, the reactive agent, the monitor agent, the facilitator agent and the user interface agent. These specific agents may be, at different levels of abstraction, multi-agent systems. The resulting architecture is shown in Figure 1.

Fig. 1. The proposed MAS Architecture for Controlling a Mobile Robot. Level 0 is the most abstract level where agents may also be MAS. Level 1 shows the MAS subsystem corresponding to the Task Planning Agent (depicted in Level 0)

Briefly, the role of these agents is as follows:

- *The Task Planning Agent* is in charge of providing a plan according to some given goal or set of goals;
- *The Reactive Agent* deals with obstacle avoidance and similar issues;
- *The Monitor Agent* tracks the whole system in order to detect, in time, (and if possible, to solve) possible failures, either in agent functioning or in the execution of the current plan;
- *The Facilitator Agent* is built into the chosen multi-agent platform and is the one which has the knowledge of the different agents that constitute the proposed architecture, where they are, and what capabilities they have ;
- *The User Interface Agent* enables the user to introduce a problem to be solved or a task to be executed by the whole system and shows the results.

In this paper we are going to focus on the Task Planning Agent. As we have said, it is a multi-agent system comprising several planning agents and a coordinator agent. The Task Planning Agent is an abstract agent in the sense that, it behaves as a task planning agent at level 0, that is, the MAS system receives a goal and returns a set of ordered actions to be performed.

The key agent in the task planner agent MAS is the Coordinator Agent, which given a planning problem, asks the planner agents for a plan or a sub-plan. The idea is to reuse several existing planners that may be suitable in certain situations. In some cases some of them may not produce a solution, or the solution might not be the best or else solutions may be provided at different speeds. The Coordinator Agent knows which planner is appropriate for each situation and asks the corresponding agent for a plan. In the case of an unexpected event, the Coordinator Agent makes requests for re-planning or even asks some planners to give a sub-plan for specific parts of the whole plan.

Hence, the aim of the work is not to create a new planner, but to use some of the well-known existing planners. As a first approach we chose as test planners, *Temporal Graph Planning (TGP)* [12], *Sensory Graph Planning (SGP)* [16], and *Prodigy* [1] [2].

As we were using existing planners, all of which have their own domain and problem definitions it would be necessary to program different agents for each selected planner. To avoid this problem we chose the Planning Domain Definition Language (PDDL) [8] [3] [4] as a standard language for encoding the planning domain. In this way, all the planners had the same domain and problem definition, making it possible to exchange planning information through agents.

3 Implementation

In this section we review the multi-agent platform used to implement the architecture of our system and the particular details of planner agents.

3.1 Multi-Agent Platform

To avoid an ad-hoc platform, we chose Open Agent Architecture (OAA) [11] as the multi-agent platform. It has certain advantages over other multi-agent

platforms including programming in C++ or in Java and the fact that it runs on Linux. Programming in C++ is important to us for two reasons: we can utilize the real-time module of Linux to program the reactive agent and we can re-use some of the existing code which, at the time, controls the mobile robot (a Pioneer 2DX).

An important thing to note is that the OAA was integrated with ARIA (from the SRI team, who developed OAA [6] [7]), and some commercial libraries that come with the pioneer [5] and which are used to program the reactive agent.

The OAA has particular agents to guarantee the correct functioning of the platform. One worth mentioning is the Facilitator agent, which provides the agent community with a number of services for routing and delegating tasks and information among agents. The role of this agent is important because it is where, upon connection, each agent registers its functional capabilities and the specifications of its public data. Moreover when a request is sent to the agent community specifying at a high level the description of the task along with optional constraints and advice on how the task should be resolved, the Facilitator agent distributes subparts of the task among agents and coordinates their global activity.

3.2 Implementation of Planner Agents

Figure 2 shows the integration of the Task Planning Multi-Agent Subsystem into the OAA.

It can be seen in Figure 2 that there are several planning agents. We have mostly used existing planners, written in various languages such as C, LISP, Prolog, etc. with different interpreters. In order to avoid having to reprogram an agent for each planner, we created a general planning agent, designed to act as an interface, or a kind of wrapper, between the existing planners and the OAA Facilitator Agent. This general agent communicates on the one hand to the planner server and on the other hand to Facilitator agent.

Figure 3 shows the architecture of the C++ agent and the links between the Facilitator Agent and the Planner Server. C++ agent and the Planner Server are considered to be the Planning Agent. The communication between the Facilitator agent and Planning Agent is in ICL (Inter-Agent Communication Language) and is specific to the OAA platform. This language is common for all the agents that are part of the MAS. In addition, internal communication among the Planning Agents is achieved through sockets (the planner server and the C++ agent may be in different computers). This communication is transparent for the community of agents.

Planner servers are programs written to run under the planner's language. In general, planner servers carry out the following tasks: 1) load the problem domain; 2) wait until it receive a problem through the socket; 3) run the planner; 4) send the resulting plan through the socket.

In the case of there being several planners, each of them has its own server and interpreter (if needed). In this way, we avoid certain problems; for instance if a server and its interpreter crashes, there are always the other servers running; and

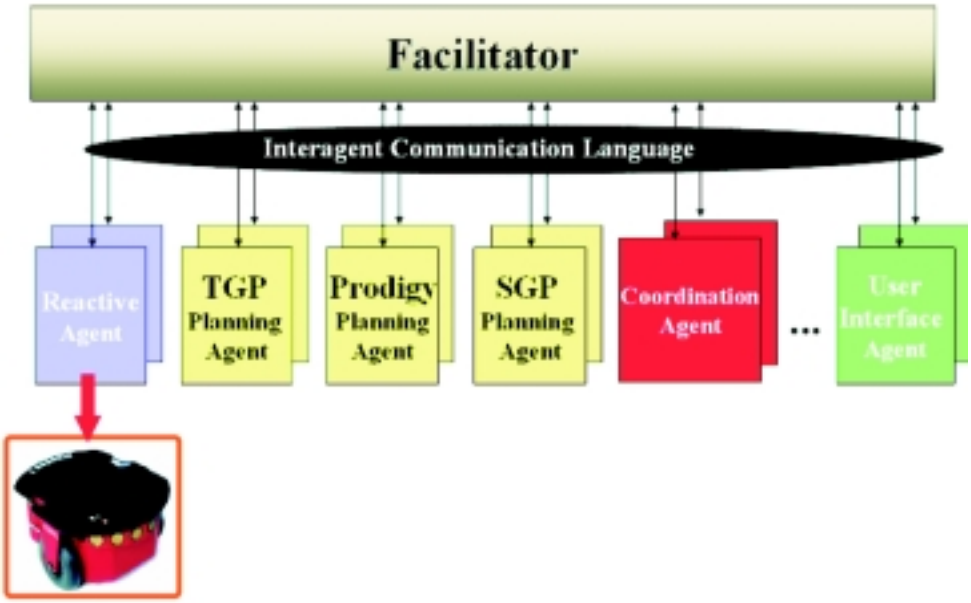

Fig. 2. The OAA Platform with the proposed implementation for the Task Planner. The figure shows the Facilitator Agent with 3 different Planning Agents and the Coordination Agent.

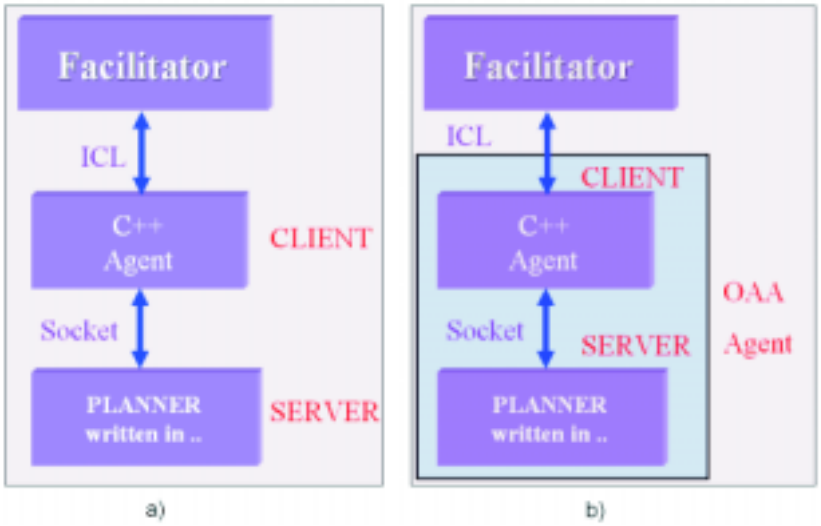

Fig. 3. a) Representation of the integration of planners into the OAA. The Facilitator agent communicates with the C++ Agent and the C++ Agent with the planner server. b)The C++ Agent and the Planner Server are considered to be the Planning Agent in this architecture.

in the case of there being different planners in the same language (for example LISP) we avoid conflicts with variables definitions.

4 Example

In order to carry out some tests, we selected *Temporal Graph Planning (TGP)*,*Sensory Graph Planning (SGP)*,and *Prodigy*. TGP and SGP are PDDL compatible and they all are written in LISP. Figure 4 shows the general OAA architecture and each planner agent with its server and interpreter.

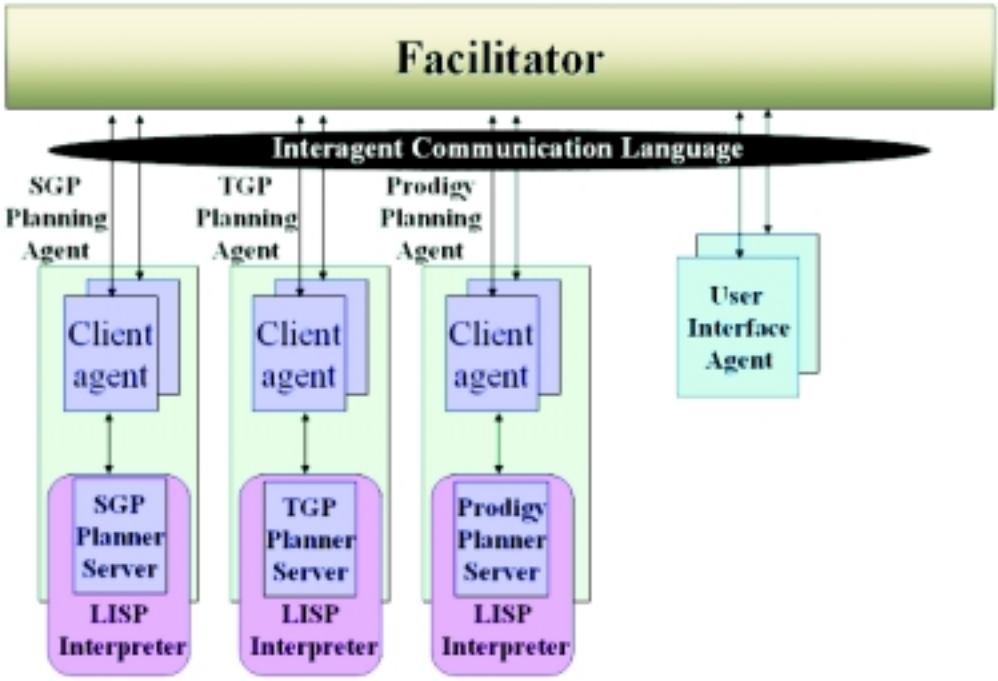

Fig. 4. General testing architecture: planning agents in the OAA with their specific planners, its servers and interpreters.

The Prodigy planner is not compatible with PDDL, so we developed a parser to translate from PDDL to Prodigy. In this way, the same domain description could be used for all of the planning agents.

The user interface agent, created for testing all the whole implementation, enables different PDDL problems to be built from a user interface and the OAA community can be asked to find a solution. As a result, three plans presented via this interface are produced.

We have tested our implementation with the *blocks world*, a well-known domain in the planning research community.

In *blocks world* there are two types of objects: a table and blocks, all of them identified by a letter from the alphabet. In its most traditional version, this universe has the following properties:

- the table can support an unspecified number of blocks,
- a block is positioned either on the table, or on another block,
- a block may have one block positioned directly above it,
- the height of a pile of blocks is limited only by the number of blocks present in the universe.

The current version of the *blocks world* domain uses five predicates :

- on(x,y): the block x is positioned on the block y;
- on-table(x): the block x is positioned on the table;
- clear(x): the block x is not covered by another block;
- arm-empty: the robot arm is empty;
- holding(x): the robot arm is holding the block x.

And uses four actions :

- pick-up(x): pick up the block x from the table;
- put-down(x): put the block x down on the table;
- stack(x,y): stack the block x on the block y;
- unstack(x,y): unstack the block x from the block y.

In PDDL these actions and predicates are defined as follows:

```
(define (domain prodigy-bw)
    (:requirements :strips)
    (:predicates (on ?x ?y)
        (on-table ?x)
        (clear ?x)
        (arm-empty)
        (holding ?x))
    (:action pick-up
        :parameters (?ob1)
        :precondition (and (clear ?ob1) (on-table ?ob1) (arm-empty))
        :effect (and (not (on-table ?ob1)) (not (clear ?ob1)) (not (arm-empty))
                (holding ?ob1)))
    (:action put-down
        :parameters (?ob)
        :precondition (holding ?ob)
        :effect (and (not (holding ?ob)) (clear ?ob) (arm-empty) (on-table ?ob)))
    (:action stack
        :parameters (?sob ?sunderob)
        :precondition (and (holding ?sob) (clear ?sunderob))
        :effect (and (not (holding ?sob)) (not (clear ?sunderob)) (clear ?sob)
                (arm-empty) (on ?sob ?sunderob)))
    (:action unstack
```

```
    :parameters (?sob ?sunderob)
    :precondition (and (on ?sob ?sunderob) (clear ?sob) (arm-empty))
    :effect (and (holding ?sob) (clear ?sunderob) (not (clear ?sob))
            (not (arm-empty)) (not (on ?sob ?sunderob))))
)
```

Given the problem defined by the **initial state** I = {on-table(A), on-table(B), on-table(C)} and the **goal state** G = {on(C,A), on(A,B)}, the corresponding description in PDDL is as follows: (define (problem simple)

```
(:domain prodigy-bw)
(:objects A B C)
(:init (on-table A)(on-table B)(on-table C)(clear A)(clear B)(clear C)
    (arm-empty))
(:goal (and (on-table B)(on A B)(on C A)(clear C))))
```

The solution to this problem is:
Pick-up(A)
Stack(A,B)
Pick-up(C)
Stack(C,A).

After submitting the problem to the planning agents, all the planners answered with the same plan for the requested task. However, they gave the answer at different speeds: TGP and Prodigy were faster than SGP.

We experimented with other well-known examples and we found that TGP was not able to solve all the proposed problems.

The useful differences between the different planners will be employed in our architecture. For example, the fact that one planner is faster than another may be desirable in dealing with complex problems: the robot can start with a simpler but faster plan and then use new and more highly elaborated plans as they are produce by planning agents. Here, the task of Coordination Agent will be very important, because it will decide how to merge plans if necessary, or how to use a part of a plan in order to solve any unforseen situations.

5 Conclusions

In this paper we presented a Multi-agent architecture used to control a single robot. The agents that make up the proposed architecture may also be multi-agent systems themselves. We described the features of the Task Planning Agent, a multi-agent system formed by planner agents and the Coordination agent. to date, we have implemented the planning agents using some existing planners.

Our preliminary results show that we are able to build a planning system by the combination of several existing planners rather than developing an optimal planner from scratch. Each problem, each situation requires different planning capabilities. So, instead of building a general planner, a MAS approach allows us to select the appropriate planning method for the problem at hand.

All our experiments have been performed in the *blocks world* domain. Our next step will be to test the planners in the robot's domain, and if needed to change or add planners. After that, we will implement the rest of the MAS architecture, and carry out experiments with the real robot.

References

1. Carbonell, J., Blythe, J., Etzioni, O., et al.: Prodigy4.0: the manual and tutorial. Carnegie Mellon, Tech. Report, CMU School of Computer Science, pp 92-150, 1992.
2. Fink, E., Veloso, M.: Prodigy planning algorithm. Technical Report, CMU School of Computer Science, pp 94-123, 1994.
3. Fox, M., Long, D.: PDDL+ : Planning with time and metric resources. Technical Report Department of Computer Science, University of Durham, UK,(http://www.dur.ac.uk/d.p.long/competition.html), 2002.
4. Fox, M., Long, D.: PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains. Technical Report Department of Computer Science, University of Durham, UK, (http://www.dur.ac.uk/d.p.long/competition.html), 2003.
5. Guzzoni, D., Cheyer, A., Julia, L., Konolige, K.: Many Robots Make Short Work. AI Magazine 18(1), pp 55-64, 1997.
6. Martin, D.L., Cheyer, A.J.,Moran, D.B.: Building Distributed Software Systems with the Open Agent Architecture. Proc. of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, Blackpool, Lancashire, UK, 1998.
7. Martin, D.L., Cheyer, A.J.,Moran, D.B.: The Open Agent Architecture: a framework for building distributed software systems. Applied Artificial Intelligence 13, pp 91-128, 1999.
8. Mc Dermott, D.: The Planning Domain Definition Language. Technical Report, Yale University, (http://ftp.cs.yale.edu/pub/macdermott), 1998.
9. McFarlane, D.C., Bussmann, S.: Developments in Holonic Production Planning and Control. Int. Journal of Production Planning and Control,vol 11, N 6, pp 5522-536, 2000.
10. Murphy, Robin R.: Introduction to AI Robotics.The MIT Press. 2000.
11. Open Agent Architecture (OAA) developer's guide, V2.2.1, SRI International, (http://www.ai.sri.com/oaa).
12. Penberthy, J.S., Weld, D.S.: Temporal planning with continuous change. Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), Seattle, WA, USA, pp 1010-1015, 1994.
13. Schillo, M., and Fischer, K. (To appear): Holonic Multiagent Systems. In Zeitschrift fr Knstliche Intelligenz, no. 3.
14. Singh, S. P. N. , Thayer, S. M.: ARMS: Autonomous Robots for Military Systems. A Survey of Collaborative Robotics Core Technologies and Their Military Applications. CMU-RI-TR-01-16, Carnegie Mellon University, 2001.
15. Uny Cao, Y., Fukunaga, A., Kahng, A.: Cooperative Mobile Robotics: Antecedents and Directions. Autonomous Robots, 4, pp 1-23,Kluwer Academic Publishers,1997.
16. Weld, D.S., Anderson, C.R., Smith, D.E.: Extending Graphplan to handle uncertainty and sensing actions. Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), Madison, WI, USA, pp 897-904, 1998.
17. Yavuz, H., Bradshaw, A.: A New Conceptual Approach to the Design of Hybrid Control Architecture for Autonomous Mobile Robots. Journal of Intelligent and Robotic Systems 34: 1-26, Kluwer Academic Publishers,2002.