Design Patterns for Combining Social and Individual Intelligences on Modular-Based Agents

Bianca Innocenti[1], Beatriz López[1], and Joaquim Salvi[2]

[1] Control Engineering and Intelligent Systems Group
[2] Computer Vision and Robotics Research Group
Universitat de Girona, Campus Montilivi, edifice P4, 17071 Girona, Spain
{bianca.innocenti,beatriz.lopez,joaquim.salvi}@udg.edu

Abstract. Design patterns have been recently concerned in the multi-agent community for the design of systems with decentralized coordination. In this paper we present a design pattern for dealing with the complexity of developing a decentralized coordination multi-agent system for controlling a single robot. In our pattern, we combine different intelligences: an individual intelligence that enables agents to achieve their own goals, and a social intelligence that makes agents understand and manage with other agents in the community. The design pattern facilitates the implementation of modular-based agents inside the multi-agent architecture and its use helps developers when incorporating new agents in the architecture. The multi-agent architecture is used to control a Pioneer 2DX mobile robot.

Keywords: agent design pattern, multi-agent system, integrated intelligence, mobile robotics.

1 Introduction

A design pattern provides a reusable solution to a recurrent problem in a specific domain [1]. A pattern does not describe an actual design, but an abstract model of the solution using specific entities of the paradigm in use. Patterns make designs more flexible, elegant, and ultimately reusable. They help designers to build new solutions without having to start from scratch [2].

Recently, design patterns have concerned the multi-agent community [1, 3], to which our research have to do with. We have developed ARMADiCo, a multi-agent architecture for a single robot and with a distributed coordination approach to share the system resources [4]. As any other kind of robot architecture, different cognitive abilities are integrated in the multi-agent approach, each requiring different artificial intelligence techniques. However, being a distributed coordination mechanism, the global system behavior emerges from individual agents (micro level behaviors). For such kind of systems, design is still an open issue [5]. One of the current proposals consist in the use of agent patterns designs [6], and we have followed such approach in the design of ARMADiCo. The agents design pattern captures common features of the agents and facilitates the incorporation of agents in the architecture.

E. Corchado, A. Abraham, and W. Pedrycz (Eds.): HAIS 2008, LNAI 5271, pp. 70–77, 2008.

Particularly, in a decentralized coordination mechanism, each agent has to deal with, at least, two kinds of intelligences: individual and social. On one hand, individual intelligence enables an agent to achieve its assigned goals (as for example, planning a trajectory for achieving a target point). On the other hand, social intelligence enables an agent to understand and manage with other agents. Consistently, any developer that wants to incorporate a new agent into the architecture has to follow the same recurrent design: define the intelligence methods to deal with individual goals, and define the methods to deal with the global robot behavior.

In this paper we present how design patterns are used in our architecture, ARMADiCo, in order to organize the different intelligences required in the agents. We describe our general agent pattern design and several instantiation corresponding to different behavioral agents. Moreover, we show how the incorporation of new agents is simple by the use of these patterns.

This paper is organized as follows. First, in section 2 the design pattern is described, together with several highlights of the ARMADiCo agents. We continue by giving some details on the behavioral agents according to the pattern. Next, the experimental set up and results are described in sections 4 and 5 correspondingly. Some related work is summarized in section 6 and we end with some conclusions.

2 Design Pattern

The proposed multi-agent architecture, called ARMADiCo –Autonomous Robot Multi-agent Architecture with Distributed Coordination-, can be described according to the main components required in classical Hybrid Deliberative/Reactive Architectures [7]. First, an interface agent is defined to interact with humans or other external agents. Second, to reason about how to achieve high level goals, we propose the mission planning, the task planning, the path planning, the battery charger and the localization agents. Third, to deal with the environment, we implement what we called behavioral agents with the following goals: go to a point, avoid obstacles and go through narrow spaces. Four, to deal with the physical world (perception and actuators), an agent is designed for each available sensor (encoder, sonar, battery sensor) and a single actuator agent has been defined (robot), due to limitations of the hardware. Finally, there are also a set of back agents that deal with other functionalities required to give support to the overall multi-agent system (e.g. Directory Facilitator).

In order to design the agents, an agent pattern has been defined. It captures common features of the agents and facilitates the incorporation of agents in the architecture. Each component of the agent pattern is designed as a module. As a consequence, our agents follow a module-based approach inside the multi-agent architecture. The current pattern is shown in Table 1. Note that each agent is different, but the pattern design offers a way to capture the different components that an agent on the architecture must have. Thus, Table 2 shows the main differences among the agents, which corresponds to the particular instantiations of their goal and coordination components (i.e. their individual and social intelligence).

Table 1. Agent pattern design

Internal State:	Mechanism used by the agent in order to know about the progress of its goals, and to update the information of the environment.
Goal:	Goal configuration: Agent goals. Goal methods: Methods that implements agent goals
Competition:	List of possible conflicting agents due to resource sharing, and list of shared resources.
Collaboration:	List of agents from/to exchange messages (request, inform).
Coordination:	Utility Computation: Method (with the required parameters) used to compute the utility value for achieving a coordination agreement Resource exchange: Method used to exchange resources from one agent to another.
Helper methods:	All supporting methods that help the agent in registering in the system, communicating, starting up, etc. They are the same for all the agents.

Table 2. AI techniques for individual and social intelligence in ARMADiCo agents

	agent	**individual (goal)**	**social (coordination)**
behavioral	goto	fuzzy	fuzzy
	avoid	pid	fuzzy
	gothrough	pid	fuzzy
deliberative	mission planning	PRS	trajectory merging
	path planning	search	-
	localization	probabilistic MonteCarlo	-
	battery charger	model based	trajectory merging
perception	sonar	probabilistic	-
	encoder	mathematical	-
	battery sensor	model based	-
actuator	robot	-	-

Regarding individual intelligence, that is, the method employed by the agent in order to fulfill its goal is specified in the *goal* slot of the design pattern. Table 2 (column "individual") shows the techniques implemented in the current implementation of ARMADiCo.

On the other hand, social intelligence in an agent is related to the interaction with other agents to resolve the resource usage. When a resource is shared by more than one agent, a conflict can arise. In order to coordinate them, ARMADiCo uses a distributed coordination mechanism, in which the agents in conflict decide which is the winner agent that takes the resource control. No central arbiter decides upon the resource usage. Since robots concerns physically grounded resources, this coordination should take into account possible disruptions in the robot behavior. For this reason the coordination process is split into two different parts: the winner determination method (utility computation slot of the agent pattern) and resource exchange method. Regarding the former, it consists of a process to assign the shared resource to the agent with the highest utility. That is, all of the agents compute an utility function for the actions they require that represents the benefit the system will receive if it carries

Agent Pattern Design		Goto Agent
Internal State		Maintain motion progress information
Goal	Configuration	Drive the robot to the goal position with the desired heading
	Methods	Fuzzy Collaborative Control System
Competition		Avoid, gothrough agents for the robot agent
Collaboration		Encoder, mission planning, battery charger agents
Coordination	Utility Computation	Based on distance to goal position
	Resource Exchange	Fuzzy-based smoothing method
Helper Methods		-

Fig. 1. Pattern Design of Goto Agent

Agent Pattern Design		Gothrough Agent
Internal State		Maintain motion into narrow places progress information
Goal	Configuration	Detect narrow places and drive the robot through them
	Methods	Model based motion
Competition		Avoid and goto agents for the robot agent
Collaboration		Encoder, mission planning, battery charger and sonar agents
Coordination	Utility Computation	Based on distance to side obstacles
	Resource Exchange	Fuzzy-based smoothing method
Helper Methods		-

Fig. 2. Pattern Design of Gothrough Agent

out the proposed action from the point of view of the agent (so their utility functions measures a gain in wealth for the whole society). The utility function is defined in the interval [0,1] for all the agents, being their values comparable. Concerning the latter, Table 2 (column "social") shows the resource exchange methods employed when the agent which wins the resource is different to the agent that has been currently using the resource up to now. Thus, robot behavior disruptions are avoided.

3 Behavioral Agents

In this section we illustrate the pattern design instances for the behavioral agents. As stated in section 2, there are three behavioral agents, the goto, the avoid and the gothrough agents. Fig. 1, 2 and 3 show the instances of our design pattern for the three behavioral agents. All of them have their individual intelligence methods (goal component of the agent pattern). The goto agent uses a fuzzy collaborative control system to move the robot to a target position, the avoid agent a PID control system to avoid obstacles and the gothrough agent a model based system to pass through narrow places.

Regarding coordination, all of them share the robot agent (resource), so conflicts could arise among them. Thus, they all have a utility computation method to determine who obtains the control over the conflicting resource (winner determination method). As stated above, the utility value varies in the interval [0, 1], 1 being the maximum value. Therefore, the agent who is controlling the resource sends to the other conflicting agents its utility value.

Agent Pattern Design		Avoid Agent
Internal State		Maintain dodging obstacles progress information
Goal	Configuration	Avoid obstacles, guaranteeing save motion
	Methods	PID control system
Competition		Goto and gothrough agents for the robot
Collaboration		Sonar,encoder, mission planning, battery charger , robot agents
Coordination	Utility Computation	Based on time to collision
	Resource Exchange	Fuzzy-based smoothing method
Helper Methods		-

Fig. 3. Pattern Design of Avoid Agent

If this value is still the highest one, the agent will continue to control the resource. Otherwise, the agent who wins the resource obtains the opportunity to use the robot agent, but it should proceed on the resource usage taking into account its impact on the physical world in a similar manner than control fusion (resource exchange method). For doing so, we propose a fuzzy method based on the information used in the coordination process that is the same for all the behavioral agents (see [8] for additional details).

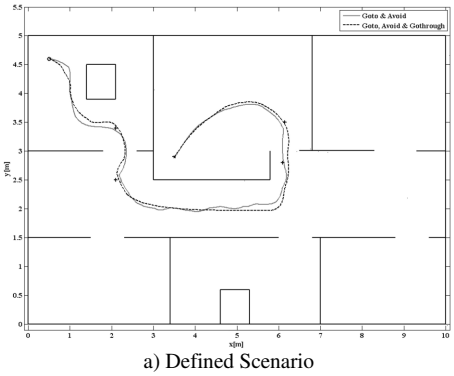
4 Experimental Set-Up

We have implemented ARMADiCo in C++ on Linux. The robot used for experimentation is a Pioneer 2DX of ActivMedia Robotics.

To test the use of the design pattern two ARMADiCo configurations have been set up, according to two different development phases:

- GA (Goto + Avoid): only the goto and the avoid agents are controlling the robot
- GAT (GA + GoThrough): the gothrough agent has been added to test the difficulty to add a new agent in the system as well as to verify that the emergent behavior of the whole system continues to be coherent and the desired one.

Next we present the defined scenario. The robot must go from Room A to Room B avoiding obstacles, as shown in Fig. 4-a).



a) Defined Scenario

Parameters	Scenario	
	GA	GAT
TD	11.64±0.14 m	11.84±0.10 m
FO	2.57±0.60º	1.56 ±0.90º
TT	71.84±6.85 s	60.17 ±3.45 s
P	99.37±0.12 %	99.31±0.21%
TG	68.64±3.14 %	99.31±0.21%

b) Results of the two tested situations

Fig. 4. Scenario to test and results of the emergent behavior of the robot

5 Results

Fig. 4-a) shows the trajectories described by the robot in grey when the GA configuration is used and in dotted when the gothrough is added (GAT). Maybe the most important issue is the fact that introducing the gothrough agent is quite simple since the agent pattern helps to determine the important aspects to consider: following it no modifications on the existing agents must be carried out.

Since our hypothesis is that the addition of the gothrough agent implies should improve the whole robot behavior, we need also to test how this happens. For this purpose, we have considered the following measures: *travelled distance (TD)*, the distance travelled by the robot to reach the goal; *final orientation (FO)*, the heading of the robot at the goal position; *total time (TT)*, the total amount of time the robot needs to achieve the goal; *precision (P)*, how closed to the goal position is the center of mass of the robot; and *time goto (TG)*, the total time the *goto* agent has the robot control.

Fig. 4-b) shows the average and standard deviation of each evaluation measure after five runnings. Comparing the results, we can see that they are very similar, except for the total time (TT) needed to arrive at the destination. With the gothrough agent, this time is decreased, meaning that the average speed is higher than when there is only the goto and the avoid agents.

6 Related Work

The application of multi-agent system to robotics has been mainly concerned to multiple robot system. For example, in [9] several soccer robots coordinate their activities based on case-based retrieval. Regarding the development of multi-agent architectures for a single robot, there are fewer works (see, for example, [10] and [11]). The main different with our architecture is that we follow a distributed coordination approach, so there is no central arbiter solving conflicts in the use of shared resources. As a consequence, we are following an emergence approach: the global system behavior emerge (macro level) from individual agents (micro level behaviors).

Regarding design issues, centralized coordination approaches use to follow a top-down traditional methodology. For distributed coordination multi-agent development, design is still an open issue. The use of design pattern has recently concerning the multi-agent community to deal with engineering emergence in decentralized autonomic system, like ours. For example, in [3] an agent pattern is proposed to encapsulate a business specific class in an AgentSpace framework and other web-based environments. In [1] the authors propose the use of agent patterns combined with workflows as a methodology for developing such emergence systems. Our design pattern is simpler than the one proposed in [1], but accomplishes the design problems we have: to state clearly the requirements of each agent, that is, an individual intelligence and a social intelligence methods. However, we should contemplate the inclusion of workflows in a future work.

Regarding the use of design patterns in robotics, there are several previous works out of the scope of the multi-agent paradigm. For example, [12] three behavioral-based are proposed to deal with three different ways of dealing with human interaction in robot control: traded, shared and supervised. In traded control, the responsibilities for producing behavior are traded between man and machine; in shared

control, an operator is guiding the robot to target, and in supervisory control the controller performs the entire task. In [13], a single design pattern is proposed to deal with the complexity of developing a robot control, according to three main layers: strategic, tactical and execution. All of these previous approaches focus on the development of a single system for controlling a robot (mainly based on object oriented methodologies) and agent patterns helps in the complex process of dealing either with real-time complexity or with human-robot interaction [14]. However, our proposal focuses on the development of a control architecture composed by a collection of agents (or autonomous systems). So although internally each agent follows a modular-architecture based on object oriented programming as well, the pattern focuses on the cooperation of the different agents in the architecture while maintaining its individual goals. Table 3 shows a comparative view of all the approaches.

Table 3. Different design patterns for robotics

Authors	Focus	Pattern(s)	Number patterns
Graves and Czarnecki [12]	human-robot interaction	traded, shared, supervised	3
Nelson [13]	real-time	Strategic+tactical+ execution	1
This paper	agent cooperation	individual+social	1

7 Conclusions and Discussion

Agents in a multi-agent architecture with distributed coordination are complex; need to deal with, at least, two kinds of intelligence (individual and social). In addition, the design of such agents is a recurrent process in which the same pattern is repeated, considering these two intelligence and design patterns offers a tool to facilitate this process.

In this paper we have described the design pattern we have employed to define the agents of the ARMADiCo robot architecture. This pattern explicitly describes the combination of different kinds of intelligences (and so techniques) at the agent level. Thus, techniques as fuzzy logic, search are used to fulfill the individual intelligence of the agent; while utility computation or fuzzy logic are used to deploy the agent social abilities. The design pattern is then implemented as a module-based architecture that conforms the agent, which in turn interacts to other similar agents in the multi-agent architecture. We have experimentally shown how the use of design patterns facilitates the inclusion of new agents in the architecture, when applying it to control a mobile robot.

As a future work, we are thinking on dealing with other methodological issues, as the ones proposed in [15], in order to deal with the emergence of the overall ARMADiCo architecture.

Acknowledgments. This work was partially supported by the Spanish MEC Project DPI2006-09370 and by the DURSI AGAUR SGR 00296: Automation Engineering and Distributed Systems Group (AEDS).

References

1. Gardelli, L., Viroli, M., Omicini, A.: Design patterns for self-organizing multiagent systems. In: Proceedings of EEDAS (2007)
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, Reading
3. Silva, A., Delgado, J.: The agent pattern: A design pattern for dynamic and distributed applications. In: Proceedings of EuroPLoP 1998 (1998)
4. Innocenti, B., López, B., Salvi, J.: A multi-agent architecture with cooperative fuzzy control for a mobile robot. Robotics and Autonomous Systems 55, 881–891 (2007)
5. De Wolf, T., Holvoet, T.: Towards a methodology for engineering self-organizing emergent systems. Self-Organization and Autonomic Informatics (I), Frontiers in Artificial Intelligence and Applications 135, 52–61 (2007)
6. Tahara, Y., Ohsuga, A., Honiden, S.: Agent system development method based on agent patterns. In: Proceedings of the 21st ICSE 1999, pp. 356–367 (1999)
7. Murphy, R.R.: Introduction to AI Robotics. MIT Press, Cambridge (2000)
8. Innocenti, B., López, B., Salvi, J.: Resource coordination deployment for physical agents. In: From Agent Theory to Agent Implementation, 6th Int. Workshop AAMAS (2008)
9. Ros Espinosa, R., Veloso, M.: Executing multi-robot cases through a single coordinator. In: Proceedings of AAMAS 2007 (2007)
10. Neves, M.C., Oliveira, E.: A multi-agent approach for a mobile robot control system. In: Proceedings of MASTA 1997 - EPPIA 1997, pp. 1–14 (1997)
11. Busquets, D., Sierra, López de Màntaras, R.: A multiagent approach to qualitative landmark-based navigation. Autonomous Robots 15, 129–154 (2003)
12. Graves, A., Czarnecki, C.: Design patterns for behavior-based robotics. IEEE Trans. on Systems, Man & Cybernetics, Part A (Systems & Humans) 30(1), 36–41 (2000)
13. Nelson, M.L.: A design pattern for autonomous vehicle software control architectures. In: Proceedings of 23rd COMPSAC, pp. 172–177 (1999)
14. Zalewski, J.: Real-time software design patterns. In: 9th Conf. on Real-Time Systems, Ulstron, Poland (2002), `http://citeseer.ist.psu.edu/zalewski02realtime.html`
15. De Wolf, T., Holvoet, T.: Using UML 2 activity diagrams to design information flows and feedback-loops in self-organizing emergent systems. In: Proceedings of EEDAS, pp. 52–61 (2007)