PROGRAMMING REACTIVE AGENT-BASED MOBILE ROBOTS USING ICARO-T FRAMEWORK

José M. Gascueña, Antonio Fernández-Caballero
Universidad de Castilla-La Mancha, Escuela de Ingenieros Industriales de Albacete
Instituto de Investigación en Informática de Albacete, 02071-Albacete, Spain

Francisco J. Garijo
Institut de Recherche en Informatique de Toulouse, Equipe SMAC
Université Paul Sabatier, 118 route de Narbone, 31062 Toulouse Cedex 9 – France

Keywords: Agents, Multi-Agent Systems, Agent-Oriented programming languages, Agent framework.

Abstract: This paper describes the experience and the results of using agent-based component patterns for developing mobile robots. The work is based on the open source ICARO-T framework, which provides four categories of component patterns: agent organization pattern to describe the overall architecture of the system, cognitive and reactive agent patterns to model agent behaviour, and resource patterns to encapsulate computing entities providing services to agents. The experimental setting is based on the development of a team of cooperating robots for achieving surveillance tasks.

1 INTRODUCTION

There is a huge amount of work and valuable proposals about agent oriented programming frameworks and languages (Bordini et al. (2005), Braubach & Pokahr (2009). This paper analyses the MAS development process using the ICARO-T framework (Garijo et al., 2008). The differentiating factor from ICARO-T is the use of component patterns for modelling MAS. Development guidelines for creating application components using agent patterns are also provided. The main advantage of ICARO-T framework is that it provides to engineers not only concepts and models, but also customizable MAS design and Java code fully compatible with software engineering standards. While other agent based platforms, such as FIPA, focus on communication standards, ICARO focuses on providing high level software components for easy development of complex agent behaviour, agent coordination, and MAS organization. An additional reason for choosing ICARO-T is cost-effectiveness of agent patterns in application development (Garijo et al., 2004).

The ICARO-T framework is the result from the cumulative experience in the development of agent-based applications in the last ten years. Therefore, the framework architecture and the underlying patterns have been elaborated, refined and validated through the realization of several agent-based applications. The first such system discovered patterns for building reactive and cognitive agents. It was a cooperative working system (Garijo et. al, 1998), which was refined with the development of a project management system for the creation of intelligent network services (Gómez- Sanz, Pavón & Garijo, 2000). Scalability of the cognitive agent model was considered in a context with thousands of users, in a MAS that supported the personalization of web sites (Gómez-Sanz, Pavón & Díaz-Carrasco, 2003), and the reuse of this solution in an online discussion and decision making system (Luehrs, Pavón & Schneider, 2003), as well as a prototype to validate the MESSAGE methodology (Caire et. al, 2002).

In 2008 a new version of the framework was delivered as open source (http://icaro.morfeo-project.org/); then new teams started using ICARO as support for academic courses and for research projects (e.g. the e-learning project ENLACE (Celorrio & Verdejo, 2007)). Utilization as implementation support for a new integrative MAS methodology is also considered (Gascueña & Fernández-Caballero, 2009a) in the domain of multisensory surveillance (Pavón et al., 2007; Gascueña & Fernández-Caballero, 2009b).

ICARO-T offers three categories of reusable component models: *agent organization models* to describe the overall structure of the system, *agent models*, based on reactive and cognitive agent behaviour, and *resource models* to encapsulate computing entities providing services to agents. The *reactive agent* architecture is made up of three components: perception, control and actuation. The perception works as an event handling mechanism. It stores incoming events, delivering them to the control on demand. The control is modelled as an extended finite state machine which consumes events stored in the perception, and performs transitions by changing its internal state and invoking actions in the actuation model. Reactive agents behave like event-consuming processes which change their internal state and execute operations according to their state transition table. An extensive description of the *cognitive agent* is available in (Pavón, Garijo & Gómez-Sanz, 2008).

The reactive agent is able to receive events from different components (agents and resources) through their use interface. The perception (1) provides interfaces to store events in a queue and to consume them, and (2) provides events to the control when requested via the perception consumption interface. To achieve their goals, the agents need to interact with the computing entities in their environment. Agents view these entities as “resources”. More formally, in agent-based applications developed using ICARO-T *resources* are those computing entities that are not agents, and are used by the agents to obtain information for achieving their objectives. *Basic computing entities* including components for building new agents and resource models are also available in the framework.

An application in the ICARO-T framework is modelled as an *organization* made up of controller components, which are *agents*, and *resources*. Therefore, there are three layers in the organization: the *control layer* (CL), which is made up of controller components; the *resource layer* (RL), made up of the components that supply information or provide some support functionality to the agents to achieve their goals; finally the *information layer* (IL) contains ontology and/or information entities needed for modelling both the framework itself and applications.

2 REACTIVE COLLABORATING MOBILE ROBOTS

The selected case study, for illustrating the development process using ICARO-T, implements the collaboration among several mobile robots (a minimum of five) to carry out a common surveillance task in an industrial estate. The robots navigate randomly through pre-defined surveillance paths in a simulated environment. When there is an alarm in a building, a robot is assigned the role of the chief, three robots will be subordinated, and the other ones will be expecting in rearguard to receive orders from the chief (e.g. to replace a damaged robot). Failures are discovered by the robot itself when any of its mounted devices (e.g., sonar, laser, camera, etc) does not work in a right way.

The robots perceive that an alarm has occurred through two mechanisms: (1) the security guard notifies robots that an alarm has occurred and where it has taken place, (2) the robot is equipped to perceive an alarm itself when it is close enough to the corner of a building; therefore it does not have to wait for the security guard announcement. The alarm is covered when a robot coalition (one chief and three subordinates) surrounds the building, that is, the robots that form a coalition are located at the four building corners where the alarm has occurred.

2.1 Application Resource and Agent Identification

The first step undertaken by the developer, in order to implement a multi-agent system with ICARO-T, is to identify the application agents and resources from the established requirements. For the proposed application an agent and three resources were identified. *RobotApplicationAgent* is a reactive agent that supports the robot functionalities. *InterfaceRes* is a visualization resource that allows the user to interact with the application (simulating an alarm rising in a given building, notifying that an alarm has occurred, simulating that a robot has detected a failure, and restarting the application), and to visualize what is happening in the simulated environment. *EnvironmentRes* is a resource that provides information about the simulated environment (industrial estate dimensions, building where the alarm has taken place, robots that do not work well, and robots initial locations). *RobotLocationRes* is a resource that stores the robots location and the moment in which they were updated.

2.2 The Application Agent Description

The reactive agent behaviour is modelled with a finite state automaton, where the states represent concrete situations of the agent life cycle. The state diagram interpretation corresponding with the reactive agent that controls a robot is as commented next. There are three kinds of states: initial (*InitialState*), final (*FinalState*) and intermediate ones (e.g., *AlarmDetection*, *Rearguard*, and so on). When an agent is in a given state and its event queue has an event, which belongs to the valid inputs (events) for transition, then the agent transits from the current state to the transition target state and executes the action associated with that transition. This mechanism is not repeated again in the new state until the execution of such an action has been completed. There is a kind of particular transition, the universal transition, which is valid for any automaton state. This transition takes place for a given input. Then, the action is executed and the automaton transits to the next state, regardless of the automaton's state. For the reactive agent to be capable of interpreting the graphically represented automaton it is necessary to express it in a textual way through an XML file. Actions are part of the reactive agent automaton and are defined as methods of semantic actions. Table 1 has been introduced to better understand how the modelled automaton works.

2.3 Application Resources Description

Application resources inherit the management interface from the resource pattern. The developer should define the use interface and the class that implements the interface for each resource. In general, an interface *NAMEUseItf* is created in a package called *icaro.applications.resources.NAME* for a NAME resource. This interface defines methods callable from other components.

2.4 How to Access a Resource and Send Events

Firstly a variable, whose type is resource use interface, is defined. After that, the resource use interface is retrieved from interface repository. Then, all is ready to access the methods offered by the interface. A resource accesses another resource by following a mechanism similar to the previously described one. Next, we illustrate how an event containing information is sent by an agent. First, an array of objects is created with a size equal to the objects to be sent, and then it is initialized. Let us highlight that the order of assignment should be the same that appears in the parameters definition of the semantic action that will be executed upon receiving the event. After that the interface repository use interface is retrieved and it is used to get the use interface associated with the agent instance to which the event will be sent. Then, the event is created. Finally, the event is sent using the use interface. Notice that sending an event to an agent from a resource follows this same idea.

2.5 Organization File Description

The application organization is described through an XML file (in our case it is "RobotSimulation.xml") that conforms to "OrganizationDescription-Schema.xsd" file. Firstly, the XML file describes the organization components features: managers' behaviour, application agents' behaviour, and application resources description. Secondly, the application instances are defined. Moreover, the instances managed for each manager are also specified. Finally, a script file in bin folder should be created to launch the developed application. This file will contain the following command: '*ant -buildfile=../build.xml run –DdescriptionPath= DESC*' (in our case DESC is RobotSimulation).

3 CONCLUSIONS

In order to cope with application development complexity, the availability of architectural framework ICARO-T facilitates the development of MAS in several ways. (1) The categorization of entities either as agents or resources implies a clear design choice for the developer. (2) Environment can be modelled as a set of resources, with clear usage and management interfaces. There are standard patterns and mechanisms in the framework to facilitate their access. (3) Management of agents and resources follows certain patterns and most management functionality is already implemented. This relieves the developer of a considerable amount of work, and guarantees that the component will be under control. (4) The framework enforces a pattern for system initialization, which is particularly important in MAS where multiple distributed entities have to be initialized consistently, and this turns out to be a complex issue in many systems. (5) Agents work as autonomous entities and encapsulate their behaviour (reactive, cognitive) behind their interfaces.

Table 1: Description of the automaton's actions.

Action	Description
move	The agent sends itself a *newStep* event if there is no alarm.
notifyPosition	Determines if the agent becomes the chief. The chief is the agent closer to the alarm and the ties are solved in favour of the agent that has a lower index. The agent sends itself a *ChiefDesignation* event if it becomes the chief.
stayInRearguard	The agent updates its role as rearguard and learns who the chief is. The robot does not move while it has this role.
subordinate	The agent (1) updates its role as subordinate and learns who is the chief, and (2) sends itself a *newStep* event.
assignRoles	The agent (1) updates its role as chief, (2) assigns to what corner the chief should go, (3) assigns to what corners the three next closets agents to the alarm should go to, and sends them a *subordinateDesignation* event that contains the following information: the target corner that it should occupy and who is the chief robot, (4) send to the other agents a *rearguardDesignation* event, and finally, (5) sends itself a *newStep* event. The *notifyPosition* action may be consulted to know how the ties are solved.
goToFreeCorner	If the chief/subordinate agent is on the assigned target corner, then it sends itself an *alarmTargetCorner* event; otherwise it determines to what corner it will move next, and it sends itself a *newStep* event.
notifyChiefOfTheError	If the agent is a rearguard, then it sends an *errorRearguard* event to the chief; whereas if it is a subordinate agent, then it sends an *errorSubordinate* event, which contains its identification number to the chief. In both cases the agent marks the controlled robot as damaged.
informChief	The subordinate agent sends to the chief agent an *alarmTargetCornerSub* event, which contains the subordinate agent identification number.
markReachedTarget CornerSub	The chief agent (1) marks the target corner that the subordinate agent has occupied, (2) increases the number of occupied corners, and, (3) notifies the user when four target corners have been occupied.
markReachedTarget CornerChief	The chief agent (1) marks the target corner that it has occupied, (2) increases the number of occupied corners, and, (3) notifies the user when four target corners have been occupied.
selectSubordinate InRearguard	The chief agent (1) increases the number of damaged robots, (2) identifies the closest rearguard agent to the target corner to be occupied by the damaged subordinate agent, and, (3) sends a *subordinateDesignation* event that contains the target corner and the chief identification number.
markFailureyRearguard	The chief agent (1) marks the rearguard agent that sent an *errorRearguard* event as damaged, and, (2) increases the number of damaged robots.
generateRoles Reasignation	The chief agent (1) marks itself as damaged, (2) increases number of damaged robots, and, (3) sends to the rest of agents a *rolesReassignation* event that contains the location of the building where the alarm occurred.

ACKNOWLEDGEMENTS

This work was partially supported by Spanish Ministerio de Ciencia e Innovación TIN2007-67586-C02-02, and Junta de Comunidades de Castilla-La Mancha PII2I09-0069-0994 and PEII09-0054-9581 grants.

REFERENCES

Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (Editors.). (2005). Multi-Agent Programming: Languages, Platforms and Applications. Springer.

Braubach, L., Pokahr, A. (2009). http://jadex.informatik.uni-hamburg.de/bin/view/Links/Agent+Platforms.

Caire, G., Coulier, W., Garijo, F., Gómez, J., Pavón, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., Massonet, P. (2002). Agent-oriented analysis using MESSAGE/UML. LNCS 2222, pp. 119-135.

Celorrio, C., Verdejo, M.F. (2007). Adapted activity deployment and configuration in a pervasive learning framework. Pervasive Learning, PL2007, pp. 51-58.

Garijo, F., Polo, F., Spina, D., Rodríguez, C. (2008). ICARO-T User Manual. Technical Report, Telefonica I+D.

Garijo, F., Bravo, S., Gonzalez, J., Bobadilla, E. (2004) BOGAR_LN: An agent based component framework for developing multi-modal services using natural language. LNAI 3040, pp. 207-220.

Garijo, F., Tous, J., Matias, J.M., Corley, S., Tesselaar, M. (1998). Development of a multi-agent system for cooperative work with network negotiation Capabilities. LNCS 1437, pp. 204–219.

Gascueña, J.M., Fernández-Caballero, A. (2009). Towards an integrative methodology to develop multi-agent systems. First International Conference on Agents and Artificial Intelligence, ICAART'09, pp. 392-399.

Gascueña, J.M., Fernández-Caballero, A. (2009). On the use of agent technology in intelligent, multi-sensory and distributed surveillance. The Knowledge Engineering Review, to appear.

Gómez-Sanz, J., Pavón, J., Díaz-Carrasco, A. (2003). The PSI3 agent recommender system. LNCS 2722, pp. 30-39.

Gómez-Sanz, J., Pavón, J., Garijo, F. (2000). Intelligent interface agents behavior modeling. LNAI 1793, pp. 598-609.
Luehrs, R., Pavón, J., Schneider, M. (2003). DEMOS tools for online discussion and decision making. LNCS 2722, pp. 525-528.
Pavón, J., Garijo, F., Gómez-Sanz, J. (2008). Complex systems and agent-oriented software engineering. LNAI 5049, pp. 3-16.
Pavón, J. Gómez-Sanz, J., Fernández-Caballero, A., Valencia-Jiménez, J.J. (2007). Development of intelligent multisensor surveillance systems with agents. Robotics and Autonomous Systems 55(12), pp. 892-903.