AGENT-BASED SOFTWARE ARCHITECTURE FOR MULTI-ROBOT TEAMS

João Frazão, Pedro Lima

Institute for Systems and Robotics
Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, Portugal
{jfrazao,pal}@isr.ist.utl.pt

Abstract: This paper describes an agent-based software architecture that intends to close the gap between hybrid systems and software agent architectures. The developed concepts and tools provide support for: task design, task planning, task execution, task coordination and task analysis for a multi- robot system. *Copywright © IFAC*

Keywords: Multi-robot systems, Agents, Software Architecture, Cooperation.

1. INTRODUCTION

In general, any software architecture should be capable of handling a family of applications. Furthermore, a non-ad-hoc architecture, based on design principles and clear concepts, allows researchers with different backgrounds to talk and share each other's experiences with less effort. Last, but not the least, it allows them to integrate their work on a larger project.
There are currently available several software tools for the mission design and development for teams of real robots like TeamBots, Mission Lab and CHARON.
TeamBots (Balch, 2002) is a collection of Java application programs and libraries designed to support multiagent systems. It supports simulation of robot control systems and execution of the same control systems on mobile robots. It includes a communication package (RoboComm), and Clay, a library to support behavior-based control systems. The simulation environment is written entirely in Java. Execution on mobile robots sometimes requires low-level libraries in C, but Java is used for all higher-level functions.
Mission Lab (Arkin, 2002) is a mission specification software that uses visual programming and reusable components. It is composed by several subsystems such as console display, a visual configuration editor, a simulator, and a runtime and usability data logging module. MissionLab generates code that runs under a distributed architecture (e.g., the main user's console can run on one computer while multiple robot control executables are distributed across a network, potentially on-board the actual robots they control.).
CHARON (Esposito and Kumar, 2002) is a language for modular specification of interacting hybrid systems based on the notions of agent and mode. It provides operations for both an hierarchical description of the system architecture (referring to the agents relations), and an hierarchical description of the behavior of an agent. The discrete and continuous behaviors of an agent are described using modes. A mode is basically a hierarchical state machine, that is, a mode can have submodes and transitions connecting them. Agents in CHARON can communicate via shared variables and communication channels. Both event-driven discrete state and time-driven continuous state system descriptions are supported.
In this paper, we describe an agent-based software architecture aiming at a considerably large set of applications involving multi-robot teams, based on simple and hopefully clear design principles that enable the development of multi-robot tasks under different architectural concepts at different levels of abstraction and by researchers with different backgrounds.

2. CONCEPTUAL MODEL

The conceptual model of the agent-based software architecture includes different types of agents that can be combined both hierarchically and in a distributed manner.
The architecture support information fusion between several sensors and the sharing of information between the *agents* by a *Blackboard* (Roth, 1985) and is geared towards the cooperation between robots.
Agents are generically organized hierarchically: "At the top of the hierarchy, the algorithms associated with the agents are likely to be planners, whilst at the bottom they are interfaces to control and sensing hardware. The planner agents are able to control the execution of the lower level agents to service high-level goals." (Esposito and Kumar, 2002).

The fundamental differences between our approach and the previous described ones are the use of a distributed blackboard for data sharing; the introduction of new agents types like the exclusive agent as well the introduction of new agent execution modes.
The elements of the architecture are the *Agents*, the *Blackboard*, and the *Control*/*Communication Ports*. Next, each of them is described in detail.

3. ELEMENTS

3.1. Agent

We define an Agent as an entity with its own execution context, its own state and memory and mechanisms to sense and take actions over the environment.
Agents have a control interface used to control their execution. The control interface can be accessed remotely by other agents or by a human operator (Henning and Vinoski, 1999). Through the control interface, an Agent can be enabled, disabled and calibrated (see the *execution modes*).
Agents share data by a data interface. Through this interface, the agents can sense and act over the world. There are Composite Agents and *Simple Agents*.

- *Composite agents* are Agents that are composed by two or more agents. The principle behind composite agents is to abstract a group of related agents. An agent society can have several types of groups. Groups represent the way that agents relate or interact with each other. Composite agents allow a group of agents to be faced as a single agent by designers, by operators or by other parts of the system. For this to be possible, a composite agent must take control over the agents that compose him. Moreover, composite agents must be easy to use: their usage should be only a matter of choosing the right type of composite agent and then plugging the controlled agents.
- *Simple agents* are agents that do not control other agents; they do not even need to know about the existence of other agents. Simple agents represent hardware devices, data fusion and control loops.

The supported agent types are:

- *Concurrent Agent*: *composite agent* that represents the simultaneous execution of two or more agents. All the agents plugged to this composite agent will execute simultaneously.
- *Exclusive Agent*: C*omposite agent* represents the exclusive execution of agents. It is used to make sure that only one of the plugged agents is executing at a given time. This is a type of agent similar to the micro-agents of the SocRob project, developed by this group (Lima and Custódio, 2002).
- *Periodic Agent* – This agent executes a given function periodically. The period is specified. This agent can be used for data fusion and control loops.
- *Sensor Agent* - A driver or a server to an hardware device of the sensor type. These are customized for each type of sensor. Usually they take data from the sensor to the blackboard.
- *Actuator Agent* - A driver or a server to an hardware device of the actuator type. These are customized for each type of actuator. Usually they take commands from the blackboard to the actuator.

The possible combinations among these agent types provide the flexibility required to build a *Mission* for a cooperative robotics project (Gamma et al., 1995). For special interactions that are not currently supported, the architecture is open to include other types of agents.
We refer to the mission as the top-level task that the system should execute. In the same robotic system, we can have different missions. The mission is a particular agent instantiation. The agents implementation is made to promote the reusability of the same agent in different missions.

3.2. Blackboard

The *Blackboard* is a distributed structure that gives support to the data exchange between the Agents. Each *entry* on the *blackboard* is a collection of *samples* ordered by their creation time. Since all the data shared between the agents goes through the *blackboard*, reads and writes are concurrent to maximize performance.

3.3. Ports

Ports are an abstraction to keep the agents decoupled from other agents. When an *agent* is defined, his *ports* are kept unconnected. This approach enables using the same agent definition in different places and in different ways. There are two types of *ports*: *control ports* and *data ports* (Figure 1).
Control ports are used within the Agent hierarchy to control agent execution. Each agent is endowed with

one upper control interface. The upper interface has two defined *control ports*.

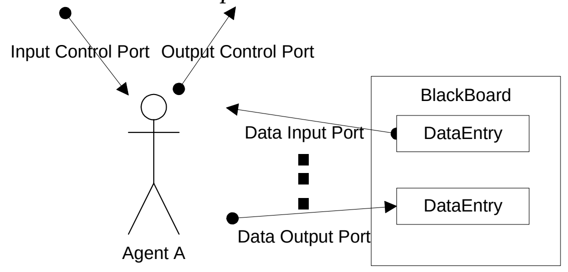

Figure 1 – Agent Control and Data Ports.

One of the ports is the *input control* port; we can see it like the request port from where the agent receives notifications of actions to perform from higher-level agents. The other port is the *output control port* through which the agent reports progress to the high level agent. This is what we denote as a consistent interface for control.

Composite agents also have a lower level control interface from where they can control and sense the agents beneath him. The lower level control interface is customized in accordance to the type of agent. For instance, an *Exclusive Agent* has as many lower level *control ports* as agents that he is controlling. An additional *data input port* is used to enable the *exclusive agent* receiving the events that select which agent to execute (Fig. 2).

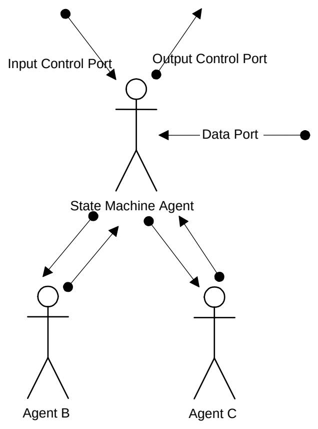

Fig. 2 – A *Composite Agent* and two controlled agents beneath him.

Data ports are used to connect the agents to the *blackboard data entries*, enabling agents to share data. More than one *port* can be connected to the same *data entry*. Several *agents* can be reading from the same place at the same time (Fig. 3). However if a *data entry* has more than a write *port* connected, some sort of contention resolution mechanism (such as in an *Exclusive Agent*) must be used.

The *data ports* are linked together through the *blackboard*. For configuration flexibility of the agent's hierarchy, the agent *ports* are not assigned in the definition of the agent.

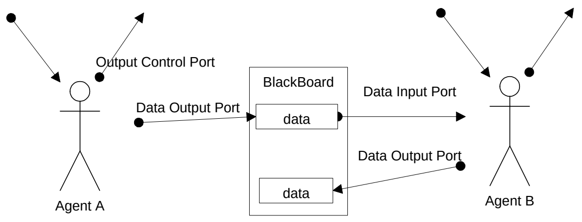

Fig. 3 – Agent A is writing a value on the Blackboard that Agent B is reading.

Ports are assigned in the instantiation of the agent hierarchy.

4. EXECUTION MODES

Traditionally, in Robotics, there is a trend towards giving importance only to the run-time impact of the robotic system architecture. Unfortunately, during several research phases, robots are stopped most of the time. Much time and resources are consumed in system design, system calibration and system analysis. These are very relevant issues often forgotten by Robotics researchers. A well-designed architecture targets the support and speed-up of these development phases.

Usually, properties such as system distribution and concurrency are relevant during the mission execution, since they provide better resource allocation and robustness.

Centralization and persistency are important properties when dealing with the robots prior to the mission execution or handling the data acquired after the mission execution. Those properties also help managing different missions for a team of robots.

Even during mission execution, system distribution is not required all the time for all the aspects. To control the robots it is better to think of them as a fleet, and to be able to exert control over the fleet from a central place, when needed.

Under this architecture, a different *execution mode* exists for each development phase of a multi-robot system.

The system hardware is composed by a central station and by the robots. The robots and the central station use a wireless network for communication.

The centralized *execution modes* of the software architecture are located on the central station. In spite of being centralized, they do interact with the robots (Fig. 4).

The *control mode* follows a distributed approach. This mode is spread across the robots (Fig. 4).

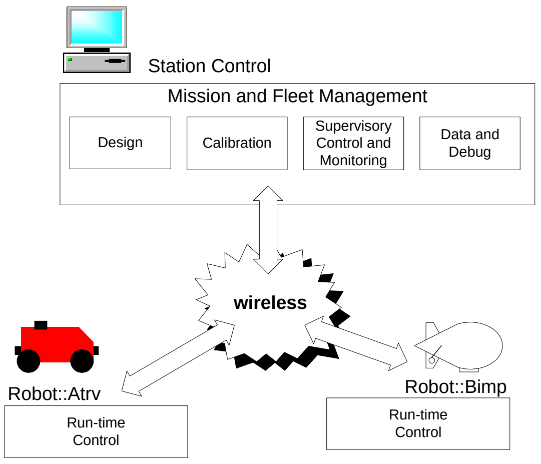

Fig. 4 – System Execution Modes – Example for the Rescue Project.

Next, we describe each of the five *execution modes* available for the *elements* described in the previous section.
First, we describe the *Control Mode* that refers mostly to the run-time interactions between the elements. Afterwards, we describe the *Design Mode*, the *Calibration Mode*, the *Supervisory Control Mode* and finally the *Logging and Data Mode.*

4.1. Control Mode

The control exerted by an upper-level agent over a lower-level agent is accomplished through special and well-defined functions: *start*, *stop*, *set* and *reset*. In this sense if we *stop* the agent that encapsulates the whole fleet, he will request his lower-level agents to *stop*, so a cascading reaction will stop all the agents' hierarchy inside each of the robots, from the top down to the lowest level hardware agents, including the robots. A similar behavior happens with the *start* command.

4.2. Design Mode

The *Design Mode* is similar to a graphics-drawing program. In these programs, there are different tools for the different graphic objects, such as lines, squares and so on (MacKenzie and Arkin, 1998).
In the *Design Mode*, instead of drawing tools for each type of graphic we have a drawing toolbox for each type of the supported agents, plus one additional toolbox for linking agents written in pure code. The output is a meta-language that represents an instantiation of the supported agents or the included code files when the agent is implemented in pure code. The language describes the connections between the agents. This meta-language is then transferred to the target robots for execution.

4.3. Calibration Mode

Usually, robots have controllers, sensory processing and hardware that must be configured or calibrated. Controllers, behaviors and perceptual processes have parameters that must be tuned. Usually this data is kept in text files for ease of modification without the need to recompile the code. For more complex calibration procedures (like color segmentation) special configuration processes must be executed sometimes.
To simplify the calibration procedure for the robot fleet, each agent has an associated calibration window, which can be requested remotely before the start of the mission. The calibration data is persistent and can be used in a later mission. To keep management of the fleet a simple job, the calibration data is stored in the central station. This data is distributed to the robots before run time.
The operator does the calibration following the instructions appearing in the remote window. For each agent involved the mission, the program asks the operator if he/she wishes to make a new calibration, to skip, to save or to load a previous one. This is done in a top-down manner. Answering *skip* to the agent that encapsulates the whole fleet will produce the result of all robots with all their agents being calibrated by the latest data used.
This mode provides support on managing the data calibration files. It also supports the way the various data types are written and read from the files.

4.4. Supervisory Control Mode

Each of the agents has, in addition to the Calibration window, an associated Supervisory Control window, corresponding to the Supervisory Control Mode, designed to be user-friendly. Therefore, the agent that controls the motors has a user-interface appropriated for its specific task. This user-interface is different from the user-interface to a (higher-level) planner agent. There are common features to all agents like the request to start, the request to stop or the request to logging. These common features are present on the associated windows and are provided automatically.
The supervisory control window uses the same program interface through which the agents receive control requests from higher-level agents and get data from the same program interface through which the agents report success, failure or progress to the

higher-level agents. The only difference is the use of a graphical window for ease of human use. If the operator chooses to control an agent from the hierarchy, the framework should disable all control requests arriving at the controlled agent from other agents.
In the supervisory control window, there is also a blackboard view. In the blackboard view, the supervisor can consult or modify the various types of variables. This is an extension of the blackboard view interface of the SocRob project (Lima et al., 2000).

4.5. Logging and Data Mode

Each of the agents can keep a logging file. If the supervisor chooses an agent whose activities are to be logged, that file is written locally inside each of the robots. After the mission ends, the log files are stored in the central station. During run-time, an operator can also choose to consult the logging of a particular agent. This mode logs, with the corresponding time tag, all the requests arriving and all reports departing an agent. Changes in the variables inside the blackboard can also be selected to be automatically logged by this mode, with the corresponding time tag. Additional logging should be made inside the code of the agent.

5. AGENT ARCHITECTURE APPLIED ON RESCUE PROJECT

Under the reference scenario for the Rescue project (Lima et al, 2003), a land robot should be able to build a topological map and be able to locate itself on that map as well as to show different navigation capabilities, such as topological navigation with obstacle avoidance. With the topological navigation the robot should be able to go from one topological state to an arbitrary topological state. It should be able to change from Topological Navigation to either Waypoint Navigation or User Operated Navigation.
The following steps describe part of the top-down instantiation of such a Rescue Mission.

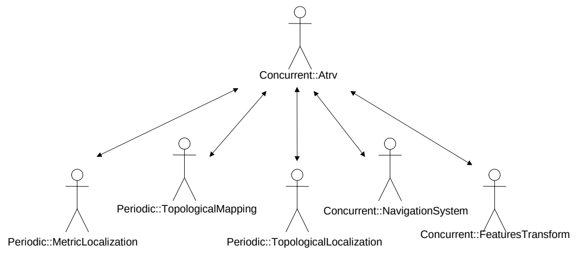
Concurrent::Atrv
Periodic::TopologicalMapping
Concurrent::NavigationSystem
Concurrent::FeaturesTransform
Periodic::MetricLocalization
Periodic::TopologicalLocalization

Fig. 5 – System Top Agent

Figure 5 shows the first system decomposition. Sensor and Actuator agents where kept out of the diagram for the ease of interpretation. In addition to one agent per sensor and one per actuator, the system is split into five main Agents. All these agents are running simultaneously, therefore they are inside a Concurrent Agent.

Each of the agents is responsible for a subsystem:

- *Features Transform* – This group of agents is responsible for picking *raw data* from the several sensors (sonar, laser, compass and image). The raw data is subsequently transformed into *features* that the Topological agents can use (Vale and Ribeiro, 2002; Vale and Ribeiro, 2003).
- *Navigation System* – This group of agents is responsible for the either the topological or the metric navigation of the robot. The Navigation sub-system includes obstacle avoidance behavior.
- *Topological Localization* – This agent gets the *data-features* and, comparing then with information taken from the *topological map*, determines where the robot is on the topological map (Vale and Ribeiro, 2002; Vale and Ribeiro, 2003).
- *Topological Mapping* – This agent is responsible for picking the *features* and building the *topological map* (Vale and Ribeiro, 2002; Vale and Ribeiro, 2003).
- *Metric Localization* – This agent is responsible for picking *raw data* from several sensors (odometric, GPS and compass). This data is fused together to determine the robot *metric position* and *velocity*.

The Fig. 6 depicts the data exchanged by the top rescue agents. The arrows represent data connections between the agents. As explained before, these connections are made throughout the blackboard. Arrows represent more than a value being exchanged. An arrow denotes that the starting agent is writing the data on *blackboard entries*. The pointed agent is reading the data from the *blackboard entries*. For ease of image reading only the *TopologicalMap* and *TopologicalPosition* data entries where represented.
If the arrow forks it means that more than one *agent* is reading the same data. When the arrow is double it means that the *agent* is reading and writing data. In the figure, the *TopologicalLocalizationAgent* is reading the *TopologicalMap* data and writing the *TopologicalPosition* data.

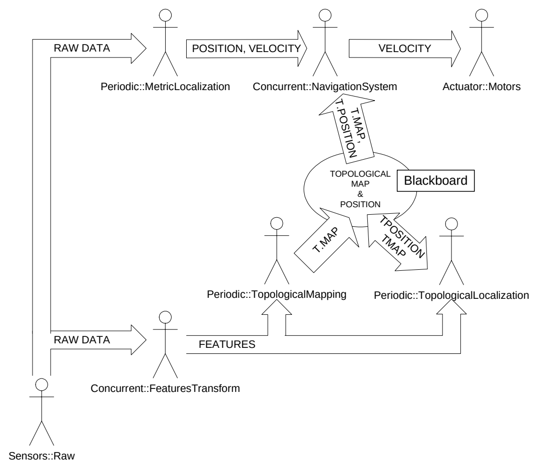
RAW DATA
POSITION, VELOCITY
VELOCITY
Periodic::MetricLocalization
Concurrent::NavigationSystem
Actuator::Motors
T.MAP, T.POSITION
TOPOLOGICAL MAP & POSITION
Blackboard
T.MAP
T.POSITION T.MAP
Periodic::TopologicalMapping
Periodic::TopologicalLocalization
RAW DATA
FEATURES
Concurrent::FeaturesTransform
Sensors::Raw

Fig. 6 – Data Flow for the first system decomposition.

All the values on the *blackboard* of the land robot are readable over the network (Henning and Vinoski, 1999). The land robot can ask the aerial robot to meet him. For now the aerial robot only has two behaviors: parametric navigation and camera target following. First the blimp tries to get to the land robot position using the blackboard land robot parametric position. If successful the blimp changes to camera target following, thus using an exclusive agent to change from one behavior to the other.

6. CONCLUSIONS AND FUTURE WORK

We have described an agent based software architecture whose key-points are:

- Agents as Reusable Software Components. Agents can be controllable and can control other agents over the network.
- Blackboard as a mean to share data over the network, addressing the problems of different agent execution rates and different agent locations.
- Several Execution modes as a mean to increase the overall system usability.

We have shown how to use the Proposed Agent Software Architecture applying it to the work already developed on the robots of our Rescue Project.
Future work on this project includes moving the *Topological Mapping Agent* to the aerial robot instead of the Land Robot and the development of cooperative navigation agents.
We also plan to use the software architecture on another R&D project where we are currently involved with a Portuguese company.

ACKNOWLEDGMENTS - Work partially supported by POSI in the frame of QCA III and by the FCT Project Rescue - Cooperative Navigation for Rescue Robots (SRI/32546/99-00).

REFERENCES

A. Vale, M. I. Ribeiro. “A Probabilistic Approach for the Localization of Mobile Robots in Topological Maps", Proc. of the 10th IEEE Mediterranean Conf. on Control and Automation, 2002.

A. Vale, M. I. Ribeiro. “Environment Mapping as a Topological Representation", 11th International Conference on Advanced Robotics (submitted), 2003.

D.C. MacKenzie and R.C. Arkin. “Evaluating the Usability of Robot Programming Toolsets" The International Journal of Robotics Research, Vol. 17, No. 4, pp 381-401, 1998.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. “Design Patterns: Elements of Reusable Object Oriented Software", Addison-Wesley, Reading, MA, 1995.

Hayes-Roth, B. “A Blackboard Architecture for Control”, Artificial Intelligence, 26:pp. 251-321, 1985.

J. M. Esposito, V. Kumar. “A Hybrid Systems Framework for Multi-robot Control and Programming”, Working notes of tutorial on Mobile Robot Programming Paradigms, ICRA 2002.

Michi Henning, Steve Vinoski “Advanced Corba Programing with C++”, Addison Wesley, 1999.

P. Lima, R. Ventura, P. Aparício, L. Custódio. “A Functional Architecture for a Team of Fully Autonomous Cooperative Robots", RoboCup-99: Robot Soccer World Cup III, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2000.

P. Lima, L. Custódio, “Artificial Intelligence and Systems Theory Applied to Cooperative Robots: the SocRob Project”, Actas do Encontro Científico do Robótica 2002 - Festival Nacional de Robótica, 2002.

P. Lima, M. Isabel Ribeiro, Luis Custodio, Jose Santos-Victor, "The RESCUE Project - Cooperative Navigation for Rescue Robots", Proc. of ASER'03 - 1st International Workshop on Advances in Service Robotics, March 13-15, 2003 - Bardolino, Italy, 2003

R. Arkin. “MissionLab: Multiagent Robotics Meets Visual Programming”, Working notes of tutorial on Mobile Robot Programming Paradigms, ICRA 2002.

T. Balch. “The TeamBots Environment for Multi-Robot Systems Development”, Working notes of tutorial on Mobile Robot Programming Paradigms, ICRA 2002.