A DISTRIBUTED ARCHITECTURE FOR CONTROL OF AUTONOMOUS MOBILE ROBOTS

P. Baroni*, **G. Guida***, **S. Mussi****, **and A. Vetturi***

*Università di Brescia, Dipartimento di Elettronica per l'Automazione
Via Branze 38, I-25123 Brescia, Italy
phone: +39 30 3715455; fax: +39 30 380014; e-mail: {baroni, guida}@bsing.ing.unibs.it

**CILEA, Consorzio Interuniversitario Lombardo per l'Elaborazione Automatica
Via R. Sanzio 4, I - 20090 Segrate (MI), Italy
phone: +39 2 26922434; fax: +39 2 2135520; e-mail: mussi@icil64.cilea.it

Abstract - *The problem of controlling autonomous mobile robots (AMR's) in real world environments has received great attention in recent years. In order to face the high complexity of this problem, a recent research trend involves the use of distributed architectures for designing and implementing robot controllers. In this paper we present a novel distributed architecture for control of autonomous mobile robots. The proposed architecture includes many autonomous agents able to communicate and to cooperate in order to achieve the global problem solution. Each agent is specialized in a facet of the AMR control problem and no agent has enough information and capabilities to solve the entire problem. Agent are decentralized and loosely coupled, and their cooperation is based on a task-sharing approach. An example of application of the proposed architecture to an instance of the homing problem is given.*

1. INTRODUCTION

The problem of controlling *autonomous mobile robots* (AMR's) in real world environments has received great attention in recent years. The features of an AMR depend on the particular type of task it is devoted to (for example, exploration of indoor or outdoor environments, production tasks in an automated factory, military missions, space or marine operations, etc.). Nevertheless, it is possible to specify some general requirements which are common to any AMR carrying out a particular task and interacting with the external environment [4], [14]. These include:

- the capability of considering several, different, possibly conflicting goals;
- real-time responsiveness to changes in the environment;
- robustness, ie. the capability of tolerating (at a certain extent) hardware or software faults;
- the capability of exploiting data coming from a high number of sensors in order to achieve a sufficiently detailed and certain perception of the external world;
- the capability of selecting the most suitable strategy for movement and operation, given the current features of the environment.

Most approaches to AMR control have been developed adopting a general purpose functional decomposition into sensing, planning, and acting. There is a vast literature on the traditional sense-plan-act approach and its variations (see for example [3], [20], [17]). In an alternative approach proposed by Brooks [5], the AMR control problem is faced adopting special-purpose task-achieving decompositions (often called behaviours). Many researchers have proposed systems which integrate these two approaches [1], [13], [16]. Still another approach to AMR control involves the use of Distributed Artificial Intelligence architectures, like *blackboard architectures* [7]. Such architectures are constituted of a number of different independent modules (called *knowledge sources*), each one able to deal with a small part of the overall control problem. The knowledge sources share a common, global working memory called *blackboard,* where input data and intermediate results about a specific control problem are collected (for example, sensor data, partial results obtained from specific knowledge sources, etc.). A scheduler determine the overall operation of the system, deciding, on the basis of the current content of the blackboard and of a given problem-solving strategy, the order of activation of the knowledge sources in front of a specific control problem. An example of application of this approach is the Ground Surveillance Robot (GSR) [11], [12].

All the above mentioned approaches are oriented towards a decomposition and distribution of the overall AMR control problem between a number of different modules, provided with knowledge and capabilities necessary to solve a portion of the problem at hand. Quoting Chandrasekaran [6], "decomposition of

processing is an absolutely basic strategy for controlling the complexity" of the control problem for autonomous mobile systems. However, in all these approaches, the control of module activity is (even if in different ways) centralized. When many different goals or several possible situations must be simultaneously considered, the centralized controller becomes a bottleneck. Moreover, the centralized controller makes the above approaches brittle and poorly appropriate to deal with real-size problems, typically large and complex.
This paper proposes a novel concept of distributed architecture that, exploiting both task distribution and control distribution, can overcome these limitations. The paper is organized as follows. Section 2 reviews the concept of distributed problem solving. Section 3 describes the proposed distributed architecture and comments on its main features. Section 4 illustrates a simple application of the proposed architecture in the solution of a particular instance of the navigation problem, namely the homing problem. Finally, section 5 summarizes the main advantages of the proposed approach, and suggests directions for future research.

2. DISTRIBUTED PROBLEM SOLVING: A DEFINITION

A simple and clear definition of *distributed problem solving* (DPS) is the following: DPS is a cooperative activity of a group of decentralized and loosely coupled modules called *agents* [19]. Agents cooperate in the sense that no one of them has sufficient information and capabilities to solve the entire problem: sharing of information and cooperation are necessary to allow the group as a whole to produce a solution. Agents are decentralized in the sense that both control and data are logically and often physically distributed: there is neither global control, nor global data storage. The agents are loosely coupled in the sense that individual agents spend most of their processing time in computation rather than in communication.

Distributed problem solvers offer, in comparison to centralized or monolithic approaches, several advantages, including: speed, reliability, extendibility, and ability to tolerate uncertain data and knowledge. Therefore, DPS naturally matches some of the main requirements of AMR's listed in section 1.

3. THE PROPOSED ARCHITECTURE

3.1 General organization

From a general point of view, our architecture is constituted - according to the general DPS concept defined in section 2. - by a collection of *agents*, able to solve a portion of the AMR control problem, and capable of cooperating according to a *task-sharing* approach. In this approach, when an agent is faced with a task too large or too complex, it may request assistance to other agents available in the architecture. It first decomposes the task at hand into more manageable subtasks and then it attempts to find other agents with the appropriate competence to handle them. Agents are supposed to be *benevolent* [9], [15], i.e. each agent is always available to cooperate to the global problem-solving activity by carrying out the assigned tasks.
Each agent has its own local problem-solving capability; it can autonomously reason on some part of the problem and can produce partial solutions. To carry out its job, an agent may resort to other agents, which are asked to solve subproblems it can not solve directly. Therefore, the global problem-solving task is performed incrementally, through cooperation among the agents.
In order to organize agents in a disciplined and effective way, on top of agents, the higher-level concept of area has been defined. An *area* is a collection of agents sharing the same competence domain. Areas are therefore obtained through a partitioning of the agent set, according to specific criteria depending on the application domain considered. Each agent belongs to one and only one area, the agents belonging to the same area are called the *near agents*. Areas have no specific problem solving capabilities; they only have an intermediary role in the management of communication - and therefore cooperation - among agents. Areas analyze requests arriving from other areas and forward them to the appropriate near agents, and analyze requests generated by near agents and appropriately address them to the relevant areas. Thus, agents can directly interact only with near agents; interaction with the other agents occurs solely through areas. The active role of an area has been modeled explicitly: each area has an associated management mechanism, called the *area manager*. For all the agents of a given area, the area manager of their area is called the *near area manager*.

The above concepts are illustrated in Figure 1., with reference to a fragment of the architecture developed for the test-case discussed in section 4.

3.2 Agents

According to the task-sharing model of cooperation introduced above, an agent must be able to carry out four functions namely:

- decomposing large or complex problems into simpler subproblems (subproblems require for their solution fewer resources than the original problem);
- allocating the subproblems to suitable near agents or, if no appropriate near specialist is found, to the near area manager;

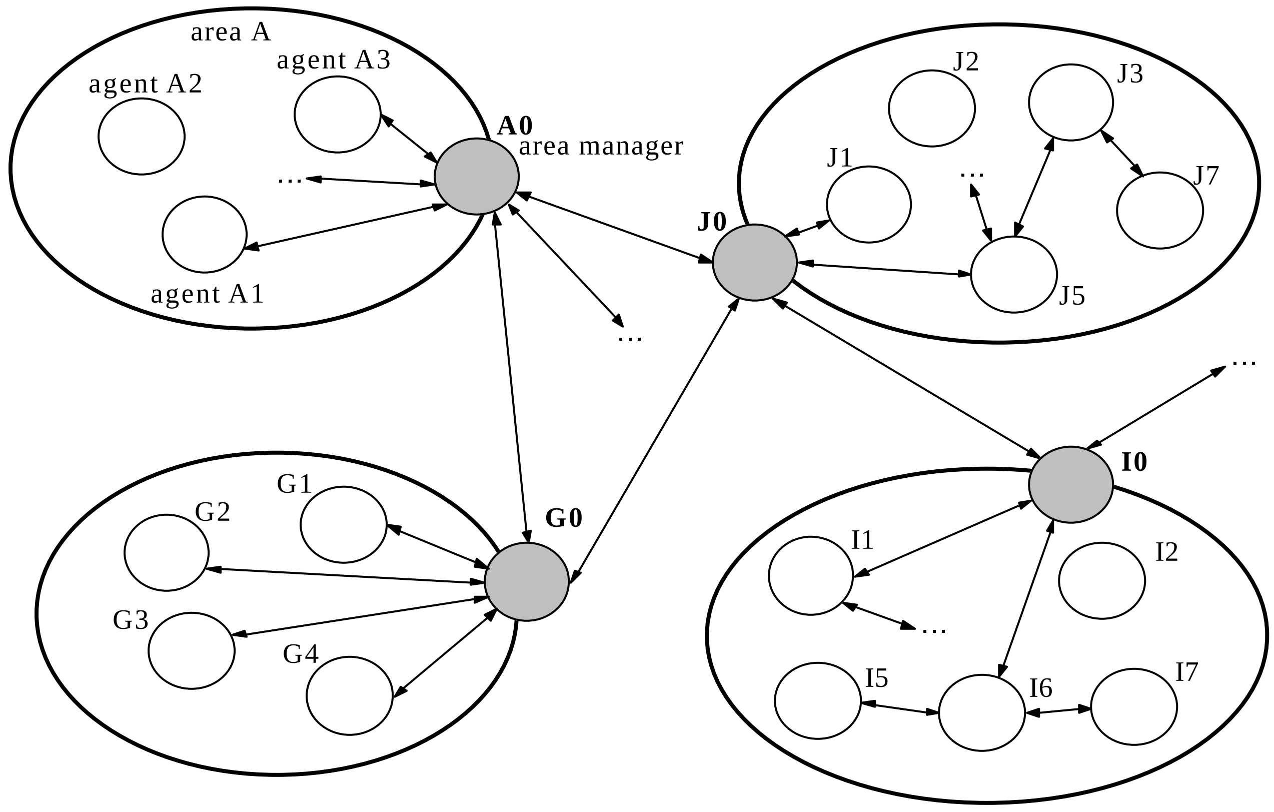

Figure 1: Overall architecture organization.

- solving subproblems belonging to its specific competence domain;
- communicating with near agents and with the area manager either for requesting their assistance or for transferring the (partial) results obtained.

In order to implement these functions, the internal structure of an agent has been organized into three modules, namely: the kernel, the mantle, and the shell.
The *kernel* is in charge of performing the specific problem-solving tasks the agent is capable of.
It contains the agent's own problem-solver, be it a procedure, a knowledge base coupled with an appropriate reasoning mechanisms, or any other problem-solving device. The internal structure and organization of the kernel is heavily dependent on the type of problems the agent is specialized to solve and on the problem-solving paradigm it is based upon. For its problem-solving activity, the kernel uses *problem knowledge*. Both the structure and the knowledge used by the kernel are totally domain dependent and different for each agent.
The *mantle* is in charge of two tasks. First, it identifies the preconditions that have to be satisfied for obtaining a meaningful solution of the problem at hand, and evaluates whether they are satisfied. In fact, in general, a problem can be successfully solved only if specific preconditions are satisfied. For instance, it is meaningless to execute a statistical procedure if only a too small set of input data is available; the results would not be significant. This task of the mantle is therefore aimed at avoiding useless effort spent at the solution of a problem that can not be solved appropriately. To this purpose, the mantle uses *condition knowledge*, that states both general principles and specific rules for identifying and verifying the preconditions for the solution of a problem within the competence domain of the agent.
Second, the mantle is in charge of implementing the fundamental problem decomposition strategy that governs the operation of every agent. It examines an incoming problem and decomposes it into subproblems, and, in turn, subproblems into new subproblems, until an appropriate problem decomposition is identified. When the solutions to all the subproblems of the decomposition will eventually be available, the mantle will compose them together to obtain the solution of the original problem. To this purpose, the mantle uses *decomposition knowledge* and *composition knowledge*, including both general strategies and specific heuristics.
The structure of the mantle is the same for all agents and sufficiently domain independent, while the specific knowledge it uses is different for each agent.
The *shell* is aimed at supporting cooperation among agents. It directly manages communication with the near agents and with the near area manager. In particular, when a problem-solving request arrives, it decides

whether it can be dealt with by the agent, i.e. whether it can be accepted or must be rejected. Moreover, when the agent generates a subproblem to be forwarded to other agents, it identifies the most appropriate near agent - or the near area manager if no near agent applies - to address it. To perform these tasks, the shell uses three types of knowledge, namely:

- *self knowledge*, i.e. (fine-grained) knowledge about the competence domain of the agent itself and the types of problems it can deal with, used to decide whether an incoming problem-solving request is acceptable or not;
- *classification knowledge*, i.e. (fine-grained) knowledge concerning the analysis and classification of incoming problem-solving requests, used to match them against agent competence and capabilities;
- *mutual knowledge*, i.e. (fine-grained) knowledge about the competence domain of the near agents and the types of problems they can deal with, used to identify the most near agent address to which a subproblem should be forwarded.

The structure of the shell is the same for all agents and sufficiently domain independent; in particular, communication occurs according to an appropriate *communication protocol* shared by all agents (and area managers). The knowledge exploited by the shell is clearly specific to each agent.

3.3 Area managers

An area manager is in charge of managing communication among the near agents and the other area managers; clearly, in supporting communication, it also deeply affects cooperation. In particular, it receives problem-solving requests from other area managers and forwards them to near agents or receives problem-solving requests from near agents and forwards them to other area managers. Its main task is the correct addressing of problem-solving requests: it receives problem-solving requests for which a suitable agent has not been identified by the near agents and must forward them to the most appropriate address (a near agent or an area manager).
To perform these tasks, the area manager uses four types of knowledge, namely:

- *self knowledge*, i.e. (large-grained) knowledge about the competence domain of the whole area and the types of problems it can generally deal with, used to decide whether a problem-solving request arriving from outside the area is acceptable or not;
- *agent knowledge*, i.e. (fine-grained) knowledge about what agents are available in the area, their competence domains, and the types of problems they can deal with, used to identify the most appropriate near agent to which a problem-solving request arriving from outside the area should be forwarded;
- *classification knowledge*, i.e. (large-grained and fine-grained) knowledge concerning the analysis and classification of incoming problem-solving requests, used to match them against area or agent competence and capabilities;
- *mutual knowledge*, i.e. (large-grained) knowledge about the competence domain of the other areas and the types of problems they can generally deal with, used to identify the most appropriate area manager to which a problem-solving request arriving from inside the area should be forwarded.

If it happens that an area manager is unable to identify the most appropriate direct addressing for an incoming problem-solving request, it resorts to broadcast addressing; thus a request arriving from outside the area is forwarded to all near agents, and a request arriving from inside the area to all area managers.
All area managers have the same structure, sufficiently domain independent; in particular, communication occurs according to an appropriate *communication protocol* shared by all area managers (and agents). The knowledge exploited by an area manager is clearly specific to each area.

3.4 Overall operation

In this section the overall operation of agents and area managers is briefly illustrated (a detailed description is beyond the scope of this paper).

Shell operation
Shell manages all communication between the agent and other entities. Its operation consists of the following functions.

- *Evaluating problem-solving requests.* When a problem-solving request is received from another agent or from the area manager the shell, using classification knowledge and self knowledge, evaluates if the problem belongs to the competence area of the agent or not. If this is the case the problem is accepted and sent to the mantle, otherwise a rejection message is sent to the agent which originated the problem-solving request.
- *Addressing problem-solving requests.* When a problem-solving request is received from the mantle as a result of decomposition activity, the shell, using classification, mutual, and self knowledge, tries to identify the most appropriate agent for the solution of the problem. If such an agent is identified, the problem-solving request is sent to it, otherwise it addresses the request to the near area manager.
- *Exchanging solutions and failure messages.* Solutions or failure messages coming from other agents are forwarded to the mantle. Solutions or failure messages produced by the kernel or by the

mantle are forwarded to the agent which originated the related problem-solving request.

Mantle operation

Mantle operation consists of the three functions described in the following.

- *Evaluating incoming problems.* The mantle, using condition knowledge, verifies if the preconditions necessary for solving a problem are satisfied (if this is not the case a rejection message is sent to the agent which originated the problem-solving request).
- *Decomposing incoming problems into subproblems.* The mantle, using decomposition knowledge, tries to decompose the problem into subproblems. If the operation is successful, then for each subproblem a problem-solving request is sent to the shell, otherwise direct solution is attempted and the problem is sent to the kernel.
- *Composing together incoming subproblem solutions.* When a solution is received from the shell, the mantle identifies the decomposition which the solved problem belongs to and verifies if all the other subproblems are solved. If this is the case, it composes together, using composition knowledge, the solutions of the subproblems and sends the solution of the original problem to the shell. If some subproblems are still unsolved the mantle waits for receiving the related solutions.

Kernel operation

The kernel receives problem-solving requests from the mantle and attempts their solution by using problem knowledge. A solution or a failure message is sent to the shell at the end of the problem-solving activity.

Area manager operation

Area manager operation basically consists of two functions.

- *Addressing problem-solving requests.* The area manager is mainly in charge of assisting near agents in addressing problem-solving requests. When a problem-solving request is received from a near agent, the area manager, using classification, mutual, and self knowledge, identifies the most appropriate area for the solution of the problem, and then sends the problem-solving request to its area manager.
- *Evaluating incoming problem-solving requests*. When a problem-solving request is received from another area manager, the area manager, using classification knowledge and self knowledge, evaluates whether the problem belongs to the global competence of the area or not. If this is not the case, a rejection message is sent to the area manager from which the request was originated. Otherwise, the area manager, using classification knowledge and agent knowledge, identifies the most appropriate near agent for the solution of the problem.

4. AN APPLICATION TO AMR CONTROL: THE HOMING PROBLEM

4.1 The problem faced

As an application of the distributed architecture proposed in the previous section, we consider a specific case of AMR control, namely the problem of navigation. In particular, we focus on the *homing problem*, i.e. the process by which an AMR moves towards a particular location on the basis of information obtained from a set of on-board sensors [2]. We assume that the AMR moves in a indoor environment. The particular environment topology is unknown to the robot, i.e. no environment map is available. The navigation is carried out by exploiting information obtained both from robot sensors and from common-sense knowledge about possible environment features provided to the AMR (again, not environment maps).

4.2 Experimental activity

The distributed architecture described in section 3 has been implemented, in very general terms, into a software system called AGENTS, written in C++ and running on a SUN Sparcstation 10 under Solaris 2.3. AGENTS exploits in a simple and effective way the possibility offered by the hardware/software environment to simulate on a bi-processor system a fully parallel execution of the activities of the various agents and area managers composing the architecture. Each agent is realized through three autonomous processes which implement the operation of the shell, of the mantle and of the kernel, respectively. AGENTS offers to the application designer a complete development environment for defining, implementing, and running a specific distributed architecture tuned to the problem domain at hand. Using AGENTS, a specific system has then been developed, called HOM-1, dedicated to face the homing problem. HOM-1 architecture encompasses 11 areas (58 agents, whose detailed description is beyond the limits of this paper), whose competences are shortly described below:

- Area A: *sensors management*
  controlling robot sensors and preprocessing the acquired data
- Area B: *mobile sensors positioning*
  positioning robot mobile sensors
- Area C: *sensors selection*
  selecting the most appropriate sensors to measure a given quantity
- Area D: *movement strategy selection*
  identifying the most appropriate strategy to perform a particular movement in the environment
- Area E: *movement implementation*
  implementing a particular movement strategy
- Area F: *actuators management*
  controlling robot actuators

- Area G: *object recognition*
  recognizing a known object on the basis of available sensory information
- Area H: *object characterization*
  identifying particular object characteristics, including size, position, state, etc.
- Area I: *environmental reasoning*
  reasoning about the environment, including objects, properties, constraints, etc.
- Area J: *environment exploration*
  management of strategies for environment exploration
- Area K: *user interaction*
  managing user dialog and identifying user needs

HOM-1 has been largely experimented with a test-case concerning an AMR capable of moving in an office space and of achieving assigned goals (HOM-1-OFFICE). The experimental activity carried out has allowed to prove the correctness of the proposed approach, to show its technical appropriateness, and to demonstrate its potentials for larger AMR control applications.

In the following section, we illustrate a simple working session with HOM-1-OFFICE.

4.3 A sample session

Let us suppose that an AMR, placed in a office, must achieve the goal of leaving the office. This goal may have been originated from a user command or may derive from the decomposition of a more complex task, for example, one of the agents of the area J may select an environment exploration strategy which involves leaving the office.

In any case, the goal of leaving the office represents a problem to solve for the agent in charge of it. If this problem does not belong to the competence area of the agent in charge of it, a search activity is started in order to identify another agent capable of solving it. The result of this search returns agent I6, specialized in the office environment, as the most suitable agent for the current problem (agent names are composed by a capital letter, corresponding to the area the agent belongs to, and by an identification number; X0 is conventionally the area manager of area X). Therefore, the goal "leaving the office" (say P.1) is allocated to agent I6.

Agent I6, using decomposition knowledge, decomposes the original problem P.1: "leaving the office" into three subproblems, namely: P.1.1: "finding the door", P.1.2: "measuring the door", and P.1.3: "going beyond door". Since these subproblems are beyond the competences of agent I6, the most suitable agents for their solution are looked for. Since agent I6 is unable to identify any suitable near agent for P.1.1, P.1.2 and P.1.3, the area manager of area I, namely I0, is charged of these problems.

For the sake of brevity, let us focus on the solution of subproblem P.1.1: "finding the door" (the other subproblems are solved in a similar way). The area manager I0 allocates P.1.1 to agent J1 (we omit, for the sake of brevity, the details of problem allocation, which involves the interaction between different area managers). The agent J1 belongs to the J area and is capable of coordinating other agents, belonging to different competence areas, to explore the office environment, looking for known objects. In this case, the object to look for is the door. Agent J1 decomposes problem P.1.1: "finding the door" into three subproblems: P.1.1.1: "image acquisition", P.1.1.2: "sonar map acquisition", and P.1.1.3: "door recognition". Later, these subproblems are allocated by the area manager J0 as follows:

- subproblem P.1.1.1: "image acquisition" to agent A1, capable of managing the camera;
- subproblem P.1.1.2: "sonar map acquisition" to agent A3, capable of managing the sonar sensor;
- subproblem P.1.1.3: "door recognition" to agent G1 which is capable of carrying out the door recognition task.

Moreover, since J1 knows that the solutions of subproblems P.1.1.1 and P.1.1.2 are relevant to the solution of subproblem P.1.1.3, it includes in the problem-solving requests P.1.1.1 and P.1.1.2, addressed to the area manager J0, the requirement of forwarding results to the agent to which P.1.1.3 is allocated.
The subproblems allocated to agents A1 and A3 (i.e., data acquisition from the camera and the sonar sensors), are not further decomposed and can be faced in parallel. They are solved by the relevant agents and the results of their activity are forwarded to agent G1.

Agent G1 belongs to the G area in charge of object recognition; in particular, it is specialized in recognizing doors. Agent G1 exploits the data acquired from camera and sonar sensors, in order to check whether the particular object it has knowledge about, i.e. the door, can be recognized in the portion of the environment in front of the robot. G1 synthesizes a solution (or a failure message) for the subproblem P.1.1.3: "door recognition" and sends it back, through area managers G0 and J0, to agent J1.

Agent J1 then examines the result produced by G1:

- If the door was recognized, agent J1 considers the subproblem P.1.1: "finding door" solved and can use this result for the accomplishment of the subsequent activity (i.e., P.1.2: "measuring the door").
- Otherwise, if no door was identified, the problem solving-activity is continued. In this case, agent J1 selects an appropriate strategy for exploring the

environment. For example, it may adopt the strategy of identifying all recognizable objects in the portion of the environment in front of the robot and, then, trying to exploit the information about the recognized objects, in order to push the exploration further.

In the latter case, J1 generates the subproblem P.1.1.4: "recognizing objects" which is sent by the area manager J0 to all agents belonging to G area, except G1. Each of them will try to recognize the objects pertaining to its competence domain, using data previously acquired from the camera and the sonar sensor, and then will send a solution or failure message back to J1, through area managers G0 and J0.

If no object has been recognized, agent J1 may formulate a new strategy, which involves interacting with agents of the B area in order to change the acquisition direction of the sensors and to explore another portion of the environment.
Otherwise, if some objects have been identified, agent J1 will then decide how to continue the exploration of the environment on the basis of the results obtained. For instance, if a window has been recognized, J1 may exploit the information about the presence of a window in a known direction in order to elaborate a more effective strategy for solving subproblem P.1.1: "finding the door". It generates the new subproblem P.1.1.5: "finding usual spatial relations between window and door" which is assigned by the area manager J0 to agent I1. Using common-sense knowledge, agent I1 infers that usually, in a room and hence in an office, door (the object to find) and window (the object found) are not located on the same wall. Therefore, if the robot moves leaving the window on the back, the search for the door might be favored. Therefore, after I1 has returned the solution of P.1.1.5: "finding usual spatial relations between window and door", agent J1 generates the subproblem P.1.1.6: "turning back to window", which is assigned to agent E3. This, using its own knowledge and possibly resorting to the cooperation of other agents if appropriate, carries out the subproblem P.1.1.6 and moves the robot to a new position. When this more favorable position has been reached, J1 continues the search for the door by generating three new subproblems, namely:

- P.1.1.7: "image acquisition" (a new instance of subproblem P.1.1.1) which is allocated to agent A1;
- P.1.1.8: "sonar map acquisition" (a new instance of subproblem P.1.1.2) which is allocated to agent A3;
- P.1.1.9: "door recognition" (a new instance of subproblem P.1.1.3) which is allocated to agent G1.

Data are then acquired from the new viewpoint, and the problem solving activity goes on in a way similar to that already described above. In this case, the door is recognized in the environment in front of robot and the problem P.1.1 "finding the door" is eventually solved.

4.4 Discussion

HOM-1-OFFICE is just a simple application example of the proposed architecture to the AMR control problem. A detailed comparison with related works is beyond the scope of this paper. We only propose here some preliminary remarks, focusing on two recent papers related to this topic, namely [8] and [18].

In [8] an architecture for navigation composed by three asynchronous heterogeneous components, namely a controller, a sequencer, and a deliberator, is proposed. This approach can be considered a first step towards the complete distribution of computation and control realized in our architecture.
The main requirements for robot control architectures identified in [8] as a conclusion of the application experience developed with the proposed approach include:

- robot control architectures should be heterogeneous: using different computational mechanisms to perform different tasks is straightforward and it works;
- robot control architectures should be asynchronous: slow computations should be performed in parallel with fast ones;
- robot control systems should be designed bottom-up.

Our architecture goes far beyond the early proposal of [8] and is powerful enough to completely fulfil the requirements stated above. In fact:

- each agent can feature a specific reasoning mechanism, tailored to its own problem-solving tasks;
- the computation processes associated to different agents are autonomous and can be carried out in parallel, if sufficient resources are available;
- system design and implementation can focus on individual agents separately and can be carried out incrementally (we will come back to this subject in section 5).

In [18] an architecture is proposed which corresponds to a hierarchical decomposition of the navigation problem in four levels. A high-level reasoning system is in charge of managing the overall robot operation and of defining high-level symbolic task to be carried out (e.g. "fetch a stick"). A global navigator accepts symbolic tasks from the high-level reasoning system and, exploiting topological knowledge, produces a sequence of more specific local tasks (e.g. "move to the bridge", "cross the bridge", "approach the stick", etc.). The global navigator interacts, in turn, with a local navigator which has competence about sensorimotor coordination

and is in charge of transforming local tasks into the appropriate sensorimotor activity. Finally, a robot controller, directed by the local navigator, manages robot sensors and actuators.
As it is quite evident, this proposal can be considered as a particular case of our approach, where agents having different competences interact through problem decomposition and allocation.
Slack [18] remarks that "past navigation research has largely ignored the use of situated knowledge to get a better grasp of the navigation problem". In our proposal, every different kind of knowledge related to the navigation problem is naturally framed in the relevant area, partitioned among agents, and exploited by them through suitable reasoning mechanisms. Moreover, while the architecture proposed in [18] enforces a particular hierarchical decomposition of the navigation problem, our approach does not impose any a priori decomposition scheme and allows agents to dynamically generate the most suitable decomposition for the problem at hand.

5. CONCLUSIONS

The paper has presented a novel architecture for the AMR control problem. The proposed architecture features a truly distributed organization which can ensure several practical advantages. It has been experimented in a case study concerning the homing problem.

Among the main technical contributions of the proposed approach, we mention:

- the capability to deal with multiple goals and multiple perspectives, crucial for obtaining an intelligent robot behavior;
- the possibility to manage a variety of heterogeneous knowledge sources, necessary to face real-size applications;
- the definition of a rational, disciplined, and general framework for embedding complex robot control knowledge.

Moreover, the proposed paradigm offers several advantages also from the methodological point of view. These concern some of the most critical tasks of the life cycle of a knowledge-based robot control system. In particular [10]:

- The process of knowledge analysis and modeling can be structured in a natural and effective way. Namely: (1) competence areas are identified first, (2) knowledge analysis sessions can then consider each area separately, exploring the global competences associated to the area, (3) finally, agents for each area are identified and analyzed through more focused working sessions.
- In the technical design phase, knowledge representation techniques and reasoning mechanisms can be tailored for each agent, which are allowed to be heterogeneous. This way, different techniques and methods can be applied for the design of different system components, thus ensuring better conceptual coupling, higher transparency, effectiveness, and efficiency.
- During system development, each agent can be built, tested, and validated independently from the others, thus favouring incrementalism and flexibility.
- Knowledge acquisition can be organized in a disciplined, modular, and structured way, thus enhancing quality and, at the same time, improving productivity and reducing costs. The area/agent paradigm, even if simple enough, can be a powerful tool in the hands of an experienced knowledge engineer: it can concretely contribute to reduce the effort required to deal with large and complex domains.
- During the operational life of the system, maintenance and extension tasks are made easier. In particular, new agents can be developed without modifying neither system architecture nor the other agents. In fact the addition of a new agent only requires updating the self and agent knowledge of the near area manager and the mutual knowledge of the near agents. Also the development of a new area poses little problems; only the mutual knowledge of the others area managers has to be updated. In this way, the extension of robot sensorial capabilities, the consideration of larger and more complex environments, and the addition of new robot tasks can be achieved in a modular and natural way.

Finally, let us mention that the proposed architecture offers specific features (for example, the explicit use of self and mutual knowledge) for implementing learning capabilities. This will be the main issue of a future research.

Acknowledgments

This work has been partially supported by the Consiglio Nazionale delle Ricerche, Progetto Finalizzato Robotica, Obiettivo PANDE.
We are indebted to Prof. Riccardo Cassinis for the support offered in the development of the test-case discussed in section 4.

6. REFERENCES

[1] R.C. Arkin. Integrating behavioral, perceptual and word knowledge in reactive navigation, *Robotics and Autonomous Systems* 6, 1990, 105-122.

[2] R. Basri and E. Rivlin. Homing using combinations of model views. *Proc. 13th International Joint Conference on Artificial Intelligence*, Chambery, F, 1993, 1586-1591.

[3] R.A. Brooks. Solving the find path problem by a good representation of free space. *Proc. 2nd American National Conference on Artificial Intelligence,* Pittsburgh, PA, 1982, 381-386.

[4] R.A. Brooks. A layered intelligent control system for a mobile robot. *Proc. ISSR, 3rd International Symposium of Robotics Research,* Govieux, F, 1985, 365-372.

[5] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, RA-2(1), March 1986.

[6] B. Chandrasekaran. Natural and social system metaphors for distributed problem solving. *IEEE Trans. on Systems, Man, and Cybernetics* SMC-11(1), 1981, 1-5.

[7] R. Engelmore and T. Morgan. *Blackboard Systems*. Addison-Wesley, Wokingham, UK, 1988.

[8] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. *Proc. 10th American National Conference on Artificial Intelligence,* San Jose, CA, 1992, 809-815.

[9] M.R. Genesereth, M.L. Ginsberg, and J.S. Rosenschein. Cooperation without communication. *Proc. 5th American National Conference on Artificial Intelligence,* Philadelphia, PA, 1986, 51-57.

[10] G. Guida and C. Tasso, *Design and Development of Knowledge-Based System, From Life Cycle to Methodlogy,* John Wiley & Sons, Chichester, UK, 1994.

[11] S.Y. Harmon, G.L. Bianchini, and B.E. Pinz. Sensor data fusion through a distributed blackboard, *Proc. IEEE Conference on Robotics and Automation*, San Francisco, CA, 1986, 1449-1454.

[12] S.Y. Harmon. The ground surveillance robot (GSR): An autonomous vehicle designed to transit unknown terrain, *IEEE Journal of Robotics and Automation* RA-3(3), 1987.

[13] M. Mataric. A distributed model for mobile robot environment learning and navigation. Technical Report 1228, MIT AI Laboratory, Cambridge, MA, 1990.

[14] D.W. Payton. An architecture for reflexive autonomous vehicle control. *Proc. IEEE Conf. on Robotics and Automation,* San Francisco, CA, 1986, 1838-1845.

[15] J.S. Rosenschein and M.R. Genesereth. Deals among rationals agents. *Proc. 9th International Joint Conference on Artificial Intelligence,* Los Angeles, CA, 1985, 91-99.

[16] R. Simmons. An architecture for coordinating planning, sensing and action. *Proc. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control,* 1990.

[17] M.G. Slack. Planning paths through a spatial hierarchy: Eliminating stair-stepping effects. *Proc. SPIE Conference on Sensor Fusion*, 1988

[18] M.G. Slack. Navigation templates: mediating qualitative guidance and quantitative control in mobile robots. *IEEE Trans. on Systems, Man, and Cybernetics* SMC-23(2), 1993, 452-466.

[19] R.G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Trans. on Systems, Man, and Cybernetics* SMC-11(1), 1981, 61-70.

[20] C.E. Thorpe. Path relaxation: Path planning for a mobile robot. *Proc. 4th American National Conference on Artificial Intelligence*, Austin, TX, 1984, 318-321.