A Possible Approach to the Development of Robotic Multi-Agent Systems

Massimo Cossentino
Consiglio Nazionale delle Ricerche(CNR)
Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
Viale delle Scienze, 90128 -Palermo- Italy
cossentino@pa.icar.cnr.it

Luca Sabatucci
Consiglio Nazionale delle Ricerche(CNR)
Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
Viale delle Scienze, 90128 -Palermo- Italy
sabatucci@csai.unipa.it

Antonio Chella
University of Palermo
Dipartimento di Ingegneria Informatica (DINFO)
Viale delle Scienze, 90128 -Palermo- Italy
chella@unipa.it

Abstract

The design of a an agent system for robotics is a problem that involves aspects coming from many different disciplines (robotics, artificial intelligence, computer vision, software engineering). The most difficult part of it often consists in producing and tuning the algorithms that incorporates the robot behavior (planning, obstacle avoidance,) and abilities (vision, manipulation, navigation,). It is frequent that the reuse of this parts is left to a copy and paste procedure from previous applications to the new one. In so doing many problems could arises. We propose a comprehensive approach for multi-agent systems oriented to robotics applications that uses a complete design methodology supported by a specific design tools and a pattern repository that interacting each other and with the designer allow the production of a coherent design that easily incorporates patterns coming from previously experienced features and automatically produces a large part of the final code

1. Introduction

In recent years, robotic systems have been used for increasingly complex activities, such as industrial applications that would otherwise involve physical risks to human operators, and those where a high degree of precision is required for complex assemblies. Performance in such complex activities requires sophisticated skills, obtained by developing articulated behaviors in response to the complex perceptual stimuli provided by the environment.

An important engineering challenge in all cases of these applications is the design of tasks and interactions among tasks that preserve the global requirements of the activity. Complex behaviors can emerge from interactions among robots, but they also arise from the interaction of basic behaviors exhibited by a single robot. In this context, the engineering design of cooperative behaviors both within and between robots is the marshaling of architectural abstractions that deal with interactions and coordination. Yet this topic of architectural design of robotic systems has only recently been addressed by the research community. Previously, the research agenda emphasized the more urgent but local issues of intelligent navigation, obstacle avoidance, vision and sensor data fusion, and so on.

With the increasing complexity of modern operating scenarios, interest is shifting toward a more global perspective on the design process for robotic systems. The starting point for design is a description of the whole robot mission. This is a formal, goal-oriented requirement for the system. The end point of the process is the generation of code in a suitable programming language. This process allows design-

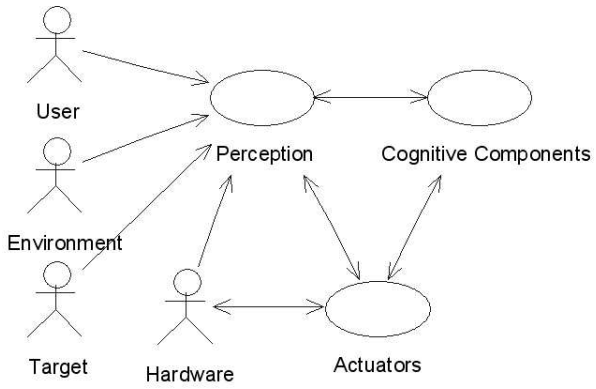

Figure 1. The architecture of a single robot from the cognitive point of view

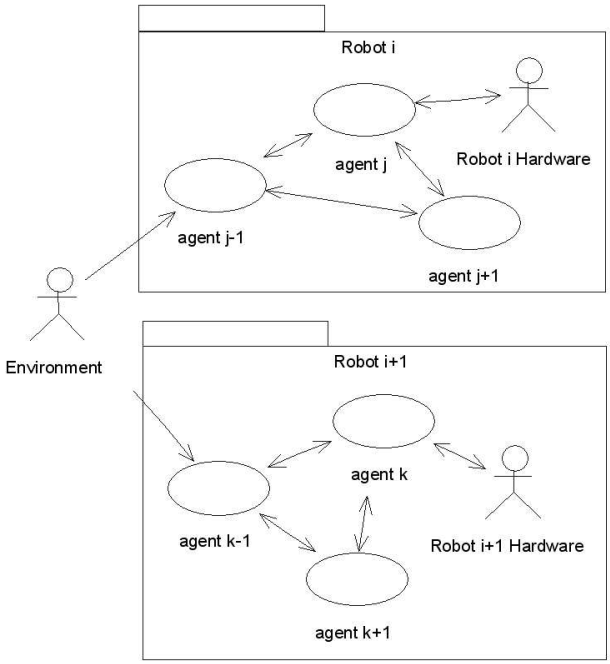

Figure 2. The structure of a generic multi-robot and multi-agent system from the functional point of view

ers to model the hardware, and regards the robot, or the robot fleet, as a system intended to satisfy some requirements thus leaving the architect free to decide if it has to be implemented in a single entity or in a collection of hardware platforms.

Designing a robotic architecture implies not only modeling the robot hardware and managing its sensor outputs, but also modeling its knowledge about the environment, and providing it with the ability to perform intelligent behaviors. Several works both from AI and Software Engineering address this topic, in particular with regards to the possibility of designing ontologies (models for bodies of knowledge about a particular domain and the relationships between them) for an agent-based system using formal descriptions that are typical of Software Engineering [10],[2]. All these approaches take UML (Unified Modeling Language) as the design formalism able to represent knowledge in a multi-agents system, due to its wide diffusion in the SE community, its status of standard graphic notation, the possibility of deriving specific ontology-oriented diagrams, and the ease to implement CASE tools.

Several other scientific works addressing the topic of the MAS (Multi-agent systems) design can be found in literature; it is possible to note that again they come from different research fields: some come from Artificial Intelligence (Gaia [19]) others from Software Engineering (MaSE [11], Tropos [4]) but there are also methodologies coming directly from Robotics (Cassiopeia [8]). They give different emphasis to the different aspects of the process (for example the design of goals, communications, roles) but almost all of them deal with the same basic elements although in a different way or using different notations/languages.

At present, agent-based architectures seem to be the most natural framework to develop a rigorous design methodology for the autonomous robots software. In fact, agents are the natural way to implement autonomous functional units that communicate using dedicated protocols and cooperate to solve complex tasks. ”Agents” may refer to the logical description of autonomous robots, or functional components or faculties within a robot. This independence from hardware and physical architecture is a necessary feature of an engineering process in which the mission of the system, or the global requirements, take priority over details about the implementation and deployment platforms.

The aim of this paper is to present a possible approach to the development of robotics applications based on the use of a design methodology (PASSI, Process for Agents Specification and Implementation [16]). For the robotic architecture we refer to the Dynamic Conceptual Spaces (DCS) theory that has been previously published by some of the authors [5]. The different design and implementation phases are supported by a specifically conceived tool (an add-in for Rational Rose) that incorporates a pattern repository allowing an high level of reusability. As a consequence, the most important results of our approach consists in the traceability of the process, the economy of time in code production and the robustness of the final code. The code generation stage uses FIPA [17] an emerging international standard for agent-based software, as the target architecture. FIPA allows very flexible architectures, not requiring the designer to employ a restricted set of communication schemas and

letting the designer delay decisions about whether and to what extent the software should be split across many hardware components.

The rest of the paper is arranged as follows. Section 2 describes in detail the DCS paradigm. In section 3, the various stages of the design process, and the motivation for our architectural choices are explained. In Section 4, the experimental setup regarding the construction of a surveillance robotic application will be reported. Finally, in Section 5, the obtained results will be evaluated and some conclusions drawn.

2. The Robotics Architecture

Our main focus is to describe a method for developing agent-based robotic systems. Consequently, in the next section, which deals with the stages of the PASSI method, we are mainly concerned with the elaboration and refinement of designs. Before describing the process, however, we need to explain the principal abstractions in terms of which such system designs are described and out of which they are composed. Our starting point is the cognitive architecture shown in Figure 1. There are three main types of components in a robotic system design, distinguished by their cognitive functions (hence their representation in functional terms as UML use cases, coherent units of functionalities provided by the system):

1. Perception, or the mapping of raw data streams into an intermediate form that is neither a raw sensory representation, nor an application-relevant symbolic representation (e.g. via sensor fusion or depth computation).

2. The cognitive faculties, such as deliberative behaviors, which we assume involve symbol manipulation. Perceptual strategies such as analysis-by-synthesis are supported by the two-way interaction between perception and cognition. Thus, cognition can direct perception by focusing attention on those external stimuli that are judged to be most relevant to the current task.

3. Actuators, which drive the robot hardware during perception tasks, and the focusing of attention. The perception-action link (by-passing cognition) allows reactive behaviors.

This architecture has already been adopted in several works [6],[7],[1]. In supporting robot system performance, the main goal of such an architecture is to go beyond the classical behavior-based model, and to provide the robot with true ”symbol grounding” capabilities due to the intermediate representation of sensory data, that is used to instantiate pieces of knowledge at the symbolic component. Through this mechanism, it is argued, the robot is able to act in a deliberative fashion more effectively.

In the present paper, however, we will take the adequacy or intelligence of system behavior and performance for granted and pay attention instead to the role of the conceptual architecture in the engineering of robotic systems. The three-way classification of agent-based functionality as perception, cognition and actuation use-cases forms the top level of a typology of possible agents. In other words Figure 1 is the highest level of abstraction in the system design, without taking into consideration the implementation details. In this context, it does not matter whether the agent instances reside in a single agent or multiple collaborating robots or other devices. In the first case, we have a potential architecture for the single robot, and in the second, we address the interaction between the external actors and the whole team in order to perform cooperative tasks.

Our approach suggests a possible abstraction from the single robot architecture to a multi robot team: the robot that is itself a multi agent system, can be viewed as a single agent in the multi robot context in which it cooperates with the others in order to reach the goals of the entire system. In Figure 2, we can see a representation of such a structure. Each robot contains several agents; some of them interact with the external environment, while others issue commands to the robot’s hardware or communicate with the agents of other robots.

While it is possible to abstract from a single agent at one level of abstraction to multiple interacting agents at a lower level, our work also supports the description of agents in terms of the tasks for which they are responsible. For design reasons, an agent is described as a colony of tasks, and these determine the role played by the agent in terms of the general architecture of Figure 1. We suppose that there is a one-to-many (but not many-to-many) relation between each one of these three areas and the agents of the system as depicted in Figure 2. Thus each agent may be classified as a perceptual, cognitive, or actuator agent, but there may be several instances of each type in a particular system.

3. The Development Process

A robotic application architecture has not been frequently developed following a rigorous and rationale process from design to code. The experimental character of these types of applications does not justify the demanding work required from the design methodology activities. Although a software engineering approach would be desirable because it would produce a well documented and structured work, it has the side effect to stretch out the time required to accomplish it.

Our proposal is to follow a methodology for developing multi-agent systems simultaneously with a large reuse of

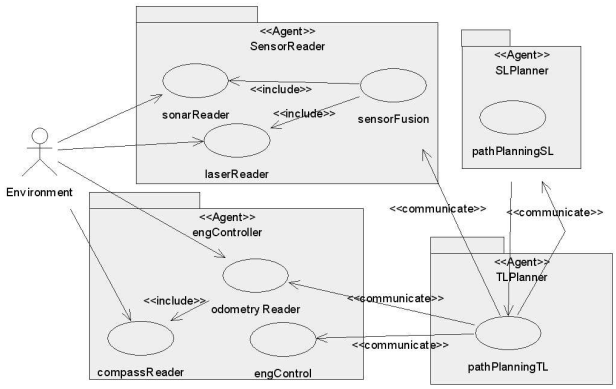

Figure 3. The part of the Agent Identification diagram related to the functionalities and relationships of the third level planner agent (TLPlanner)

design models and implementation code supplied by the use of patterns of agents.

3.1. Our Design Approach

A robotic application architecture has not been frequently developed following a rigorous and rationale process from design to code. The experimental character of these types of applications do not justifies the demanding work required from the design methodology activities. Although a software engineering approach would be desirable because it would produce a well documented and structured work, it has the side effect to stretch out the time required to accomplish it. Our proposal is to follow a methodology for developing multi-agent systems simultaneously with a large reuse of design models and implementation code supplied by the use of patterns of agents. A robotic architecture can be well modeled using a multi agents system and particularly the agent autonomy is very important because it gives us the opportunity of easily partitioning the architecture in different components, each of them independent by the others and responsible for only a portion of the overall system behavior/functionalities.

In our approach we use PASSI [16], that is a step-by-step requirement-to-code methodology for developing multi-agent software that integrates design models and philosophies from both object-oriented software engineering and MAS (Multi-Agent System). It is composed of five models that address different design concerns and twelve steps in the process of building a model.

In PASSI we use UML as the modeling language and its extension mechanisms (constraints, tagged values and stereotypes) to facilitate the customized representation of agent oriented diagrams without requiring a completely new language. Moreover, like other UML-based methodologies, PASSI is supported by a CASE (Computer Aided Software Engineering) tool, called PTK (PASSI toolkit) that is an add-in for Rational Rose. The use of PTK allows the automatic composition of some of the diagrams (built upon the information provided in the previous parts of the design), a consistency check of the work and generation of the agent code also reusing pieces coming from a pattern repository.

In the first PASSI model (**System Requirements**) the designer analyzes the system requirements and produces a decomposition of them among the identified agents. This model involves four steps:

- **Domain Description** (D.D.), in which the developer describes the functional requirements of the system using conventional use-case diagrams.
- **Agent Identification** (A.Id.), where the functionalities are assigned to different agents. As an example we can consider, in Figure 3 the part of the A.ID. diagram showing the functionalities of the third level planner agent (TLPlanner) and its relationships with the others. The functionalities of the system are represented with use cases (ovals in the figure) and the relationships among this pieces of external behavior of the system can be of three different types: include if one functionality always needs services by another to accomplish its duty, extend if another use case concurs in the solution only under precise circumstances, communicate if two use cases belonging to different agents interact to exchange data or services (this is different from standard UML where only relationships among actors and use cases can be of the communicate type). In Figure 3 we can see that the TLPLanner agent includes the third level planning functionality (pathplanningTL use case) and that in order to fulfill its mission it asks for data coming from the SensorFusion use case (that is one of the functionalities of the SensorReader agent) where readings from sonars and laser are combined in the histograms of the VFH+ algorithm. The TLPlanner agent also receives suggestions about the path to be followed by the second level planner (SLPlanner) agent and sends commands to the robot motors communicating with the engcontrol use case of the EngController agent. The feedback from odometry is then used to refine the movement commands and if an obstacle is detected and the path changed to avoid it, this information is sent to the second level planner (communicate relationship from the TLPlanner agent to the SLPlanner agent) that recalculates its path.
- **Role Identification** (R.Id.), consisting in the use of sequence diagrams to explore each agent's responsibili-

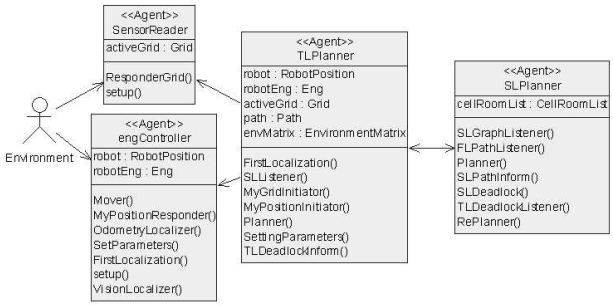

Figure 4. The portion of the Multi-Agent Structure Definition diagram (MASD) representing the agents shown in Figure 3

ties through role-specific scenarios. For example these diagrams are used for representing the interactions that the TLPlanner agent establishes in order to obtain the data required for path planning.

- **Task Specification** (T.Sp.), where we use activity diagrams to describe the capabilities of each agent. Here we also design the policy the agent uses to fire its behaviors. BDI or statechart based agents produce very different results in this phase.

The second PASSI model is **Agent Society**: it is the representation of agents' interactions and dependencies from the social point of view. Developing this model involves three steps:

- the **Ontology Description** (O.D.) is one of the typical steps of designing an agent-based system. We use class diagrams and OCL (Object Constraint Language) constraints to describe the knowledge ascribed to individual agents and their communications in terms of content language (SL, RDF, KIF [12]), piece of ontology referred and agent interaction protocol.

- **Role Description** (R.D.) consists of class diagrams showing the distinct roles played by the agents, the tasks involved in these roles, the communications and inter-agent dependencies.

- In the **Protocol Description** (P.D.), the agent interaction protocol is described using UML sequence diagrams in terms of speech-act performatives like in the AUML approach [18].

The third PASSI model is **Agent Implementation** where the architecture of the multi agent system is defined in terms of classes and methods. This model involves the following steps:

- The **Agent Structure Definition** (A.S.D.) is a classical representation of the system in terms of class diagrams. We use two different levels of abstraction producing a multi-agent diagram (M.A.S.D.) where each agent is represented by one class and the operations of this class are the tasks of the agent and one different single agent diagram (S.A.S.D.) for each agent. In the S.A.S.D. the agent main class and the tasks are represented as different elements each one with its 'real' methods that will be coded. For instance we can consider in Figure 4, the portion of the Multi-Agent Structure Definition diagram (MASD) representing the agents shown in Figure 3. The knowledge of each agent is described in the class attribute compartment (the upper one), therefore we can see that the TLPlanner agent knows the robot position (robot piece of knowledge referring to the RobotPosition concept of ontology), the grid decomposition of the room (activeGrid piece of knowledge that relates to the Grid concept) and so on. The tasks (i.e. the elementary pieces of behavior of the agent) are described in the class operation compartment (the lower one). The TLPlanner agent has a FirstLocalization task used to require the self localization to another agent, the SLListener task receives the communications from the SLPlanner agent and so on.

- The behavior of the each single agent and of the whole society is modeled in the **Agent Behavior Description** (A.B.D.)diagrams. They are activity diagrams or state charts again used at the multi or single agent level of abstraction

The fourth PASSI model is the **Code Model**. In this phase we produce the solution at the code level performing the following steps:

- In **Code Reuse** (C.R.), we take existing patterns from a repository and use them in order to fill the inner part of the methods

- During the **Code Completion** (C.C.) the source code of the target system is completed by the programmer and the final version of the software is released.

The **Deployment Model** is the last step of the PASSI process. It is a model of the distribution of the parts of the system across hardware processing units, and of their migration between processing units. It involves one step, the **Deployment Configuration** (D.C.) where we use UML deployment diagrams to describe the allocation of agents in the available processing units and any constraints on migration and mobility.

Amount of work (months)	18
Different types of agents	16
Total number of classes	230
Lines of code	10610

Table 1. The dimension of the surveillance robotic application (time spent is without pattern reuse)

3.2. Coding with patterns reuse

In PASSI great importance has the reuse of existing patterns. We define a pattern as set of different representation of the same structural/behavioral part (a couple of agents interacting in order to accomplish a cooperative goal, a single complete agent, a task of an agent or even a piece of a task) of the multi-agent system. Therefore, in our approach, each pattern is composed of a model of the structure of the involved elements (an UML class diagram), a model of the dynamic behavior (an UML activity diagram) and the implementation code [9].

The PTK (PASSI ToolKit) tool comprehends a repository of patterns and during the design process the user can select the desired patterns from a list and can import them in the project. This operation easily enriches the current multi-agent system with the functionalities or behaviors defined in the reused pattern. The repository of patterns has been developed in the AgentFactory project funded within the Agentcities initiative (www.agentcities.net).

This process brings to drastically lowering the cost of developing a multi-agent application without limiting the choice of the implementation platform. In order to prove this assertion we decided of simultaneously supporting both the JADE and FIPA-OS platforms. These are very diffused FIPA-compliant platforms that cover a great percentage of installed systems. If the designer wants to produce a JADE system, he has the opportunity of reusing the same patterns available in FIPA-OS. This is possible because the description of the pattern uses XML as a meta-language platform-independent representation (meta-pattern).

This high level description of a pattern has been introduced to separate its structure from the implementation platform. For example in the FIPA-OS platform, a task is a class that extends the Task super-class and contains a startTask() method; in the JADE platform a task is a class that extends the Behavior super-class and contains an action() method. These structural differences can be handled with a unique meta-description using high level concepts like TaskShell or TaskSetup(). TaskShell is the super-class extended from any task, while TaskSetup() is the method called when a task is scheduled.

From the meta-pattern, applying an XSLT transformation (a transformation used to change the structure of an XML document), we deduct the platform (FIPA-OS or JADE) specific static structure and dynamic behavior. This is a pattern that can contains attributes and methods compatible with the specific agent platform. For example the TaskSetup() becomes startTask() in the FIPA-OS transformation while action() in the JADE transformation.

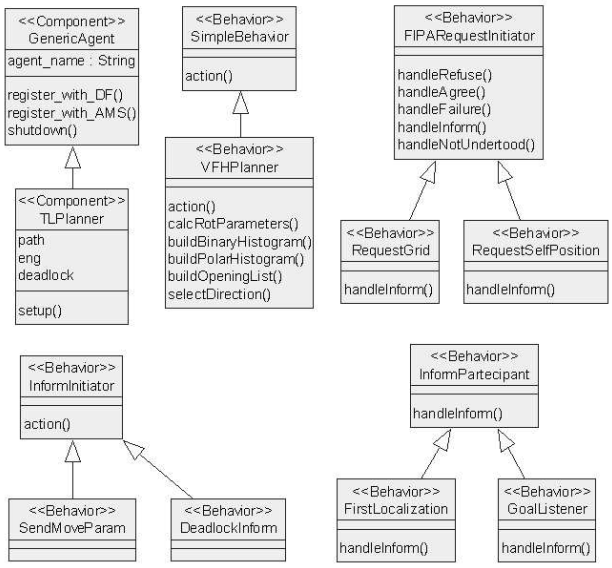

Figure 5. The structure of the VFH_TL_Planner pattern that offers VFH+ planning capabilities.

Now, the static and dynamic description of the pattern together with an additional XSLT transformation that introduces some of the implementation features of the platform, contribute to generate the JAVA code of the agent. At this skeleton we add the body of the methods (when available for the specific environment, we call these parts action patterns) obtaining a class that is complete both in the structure and in the inner code. As a consequence automatic code generation percentage grows-up with the number of pattern used in the project.

It is useful to specify that while several UML-based CASE tools can generate code, they do not have specifically conceived structures (agent, task and other specific base classes) nor they can produce significant parts of the inner code for the methods.

4. Experimental Setup

The experiment we refer in this paper consists in a robotic system devoted to surveillance tasks. More in detail the implemented functionalities comprehend the recon-

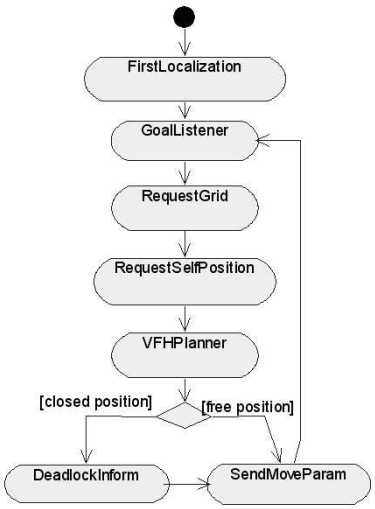

Figure 6. The sequence of tasks and the control logics of the VFH_TL_Planner agent pattern described with an activity diagram

naissance of the building, the detection of new objects in the environment (for example a bag that someone has forgotten) with the consequent update of the environmental knowledge and map description, the automatic detection of an intruder, the pursuit (and encirclement if more robots are available) of the intruder.

From the hardware point of view, the system is composed of one (but more is possible) B21 mobile robot with a computer and a stereo camera aboard; some fixed cameras are positioned in the environment (a floor of our department) in order to detect the intruder and four fixed workstations are used for agents deployments.

The software aspects are characterized by a multi-agent system implemented with a FIPA-compliant platform (JADE). The dimension of the whole project (as summarized in Table 1) are quite interesting since more than 10 thousands of lines of code and 16 different types of agents have been produced (some of them with several instances at runtime). Each one of the three different categories (perception, cognition and actuation) of our robotic architecture in the proposed experiment includes several agents. A particular effort has been dedicated to the vision subsystem that introduces a multi-level architecture allowing the dynamical introduction of new hardware (cameras) and services (agents performing different kinds of filtering and images manipulations) [13]. One instance of a grabber agent is bound to each camera (the type is FixedCameraGrabber if the camera is a single camera or StereoCameraGrabber if the camera is a stereo system) to capture the images. Several instances of these agents are used and therefore an HardwareManager agent is necessary to allow other agents to interact with the best positioned (or more useful) camera for each specific purpose. The captured images are then manipulated by other agents that can perform tracking, motion detection, camera calibration and other operations. The SelfLocalizator agent localizes the robot in the environment using two images captured by its stereo camera while looking at a landmark whose position is known *a priori*. This information is also used to correct the odometry error. Other sensors (infra-red, laser range finder and compass) are managed by an unique agent, the SensorReader.

Navigation and path planning have been realized using a three level planning approach:

The first level looks at the environment (the building) as a graph of interconnected rooms, the FLPlanner agent deals with this and brings the robot from the actual position to the room where the intruder has been found.

The second level deals with building the path inside the single room.

The third level, planning (TLPlanner agent), is related to the obstacle avoidance. It uses the VFH+ algorithm [3],[14],[15] to perform the sensor fusion and change the trajectory in order to avoid unexpected obstacles. The actuation of the movement is responsibility of the EngController agent that converts the path (composed of a series of direction commands) in the directives used to control the robot's motors.

4.1. An example of pattern of agent

Our repository of patterns include many elements specialized for robotics. One of these is the VFH_TL_Planner pattern whose functionality is planning with the VFH+ algorithm.

The VFH (vector field histogram) is a largely adopted planning technique [3],[14],[15]; it grants a fluid motion of the robot and a simple data fusion of sensors information coming from different sources.

Because of the frequent application of this algorithm, its availability as a pattern to reuse could be useful in designing a robotic application.

The static structure of the VFH_TL_Planner pattern is represented in Figure 5 where the different tasks and the base agent class are obtained specializing more general patterns (super-patterns).

The TLPlanner pattern represents an entire agent (for this reason we mark it with the *component* stereotype) including its necessary tasks. The main agent class is built extending the GenericAgent behavior that provides the ability of registering itself to the agent platform (Directory Facilitator and Agent Management System services). The VFHPlanner task (like other tasks characterized by the Behavior stereotype) that implements the VFH algorithm is

```
<Pattern name="VFHTLPlanner">
   <Agent name="TLPlanner" extends="GenericAgent">
      <Task name="VFHPlanner"
               extends="SimpleBehavior">
         <TaskSetup>
            <Code>elaborate_path@VFHPlanner</Code>
         </TaskSetup>
         <Method name="buildPolarHistogram" type="Histogram">
            <Argoment name="activeGrig" type="Grig"/>
            <Argoment name="selfPosition" type="Point"/>
            <Code>buildPolarHistogram@VFHPlanner</Code>
         </Method>
         ...
      </Task>
      ...
   </Agent>
</Pattern>
```

Figure 7. The XML meta-language representation of the TLPlanner agent

specialized from the SimpleBehavior task (one of the base and simplest behaviors). The other six tasks (RequestGrid, RequestSelfPosition, SendMoveParam, DeadlockInform, FirstLocalization and GoalListener) are all devoted to communications and are derived from pattern that refer to the use of some specific agent interaction protocols (like Request or Inform) and different roles in the communication (initiator or participant).

The main agent class has also the role of coordinating the tasks flow of control accordingly to the specifications of the activity diagram in Figure 6. In this diagram each rounded angle rectangle represents a task of the pattern.

The first activity performed by the agent is to request to another of updating the robot current position estimation (FirstLocalization task). When this auto-localization is complete the TLPlanner agent is ready to receive the plan produced by the higher level planning agent (GoalListener task). Again looking at the structural diagram of Figure 5, we can see that the task is a communication task and that it will use the Inform agent interaction protocol and the agent will play the role of participant in the communication.

When the target position (and subsequent plan) is received then the agent begins a loop: it sends the request for the active grid to the sensor fusion agent (RequestGrid task) and the request for the self position is sent to the engController agent that can read the odometry sensors and estimate the requested coordinates. With this information available the agent can execute the VFH+ algorithm (VFHPlanner task). If it is possible to move towards the goal (eventually avoiding an obstacle) the agent sends the commands to the motors (SendMoveParam task) otherwise it informs the second level planner agent that there is a stall condition and plan need to be changed (DeadlockInform task).

The static structure of VFH_TL_Planner pattern, represented in the class diagram of Figure 5 can be represented in form of a platform independent XML-based meta-language as shown in Figure 7.

This code incorporates the parts of the pattern structure and behavior that are common to the two different FIPA compliant platforms that we use: FIPA-OS and JADE.

We can see that the main agent class (TLPlanner) is reported in the agent name tag where the extension relationship with the GenericAgent pattern is specified (we use the GenericAgent pattern to introduce fundamental capabilities in more complex agents). The VFHPlanner task is shown together with some of its methods (particularly the setup method will be discussed in the following). The remaining part of the XML code is neglected because not important in this context.

From this still high level of representation of the agent implementation we deduce (with an XSLT transformation) another XML-based stage that is localized to the specific platform (for example it includes the default method of each that is startTask for FIPA-OS and action for JADE). This process could bring to the automatic generation of the complete agent skeleton (results of this kind are obtained by almost all the object-oriented CASE tools). We go beyond this step trying to obtain a great amount of code for the inner parts of the methods. This is possible if the behavior of the specific part of code is known (as in this case) and therefore it could be written for the two different platform, stored in a repository and reused when necessary. As an example consider the elaborate_path@VFHPlanner action pattern (an action pattern is a portion of code reusable inside a method) that is present in the setup method of the VFHPlanner task, at the code tag. When the XSLT transformation that produces the final JAVA (JADE or FIPA-OS) code is applied, the elaborate_path@VFHPlanner action pattern is substituted with the code represented in Figure 8 (for JADE). In the case of the FIPA-OS platform the code

```
// build polar histogram
Histogram polar=buildPolarHistogram(grid, robot_position);
// build binary histogram
Histogram binary=buildBinaryHistogram(polar);
// search open directions
LinkedList openings = buildOpeningList(binary, target_dist);
// select the best direction
int select_dir = selectDirection(openings, target_dir);
// compute the engine parameters relative to the selected direction
Params params = calcRotParameters(select_dir, target_dir);
// send the parameters to the engine agent
addBehavior(new SendMoveParam(params) );
```

Figure 8. The elaborate_path@VFHPlanner action pattern (inner part of the method) used in the VFHPlanner agent pattern

would be almost the same but the last line should be:

```
startTask(new SendMoveParam(params));
```

This shows that constructing the action pattern necessary for both the two platforms that we selected is not an impossible effort since they share the same language (JAVA) and there are little difference among them.

5. Experimental Results and Conclusions

The robotic application we presented has been originally realized using a precise design process (PASSI) but without the availability of the pattern repository. Subsequently at the introduction of this feature in the PTK tool we rebuilt the application.

About these experiments we should consider that the study and tuning of the algorithms used for vision, navigation, planning and the realization of the drivers for controlling the robotic hardware required 11 man/months; this part of the work was not repeated in the second experiment. For this reason we do not include this time in the comparison of the results that are reported in Table 2.

The code reuse percentage for the agent that extends the VFH_TL_Planner pattern, reported as an example throughout the paper, is about 31%.

This means that while the complete agent code is composed of 486 lines, after the application of this pattern to the project, the programmer reused 152 lines of code and manually added the remaining 334 (algorithmic parts of the agent, related to this specific problem and that are not present in the repository). Almost an half of the automatically produced lines of code (70/152), are lines of the inner part of the methods and not simple skeletons.

From the reuse point of view this agent can be considered a typical example. In this experiment we had agents with up to 82% of automatically generated code (SensorReader) and agents with a lower percentage (SLPlanner, 18%).

The overall percentage of automatically generated code is about 26% and the 46% of this code is method code (not skeletons). It should be considered that a larger part of the code that the designer added manually was strongly algorithmic (therefore not well suited for pattern reuse) and in both our two experiences in building the application it derived from the previous activities of alghoritms studying and tuning. This justify the differences in the coding and testing activities in the two experiments (with and without patterns, see Table 2).

In this considerations we should also add the time that the designer saves when introducing a pattern. In fact in so doing he obtains the reuse of the related portions of design.

We consider valuable this process not only for the resulting high productivity but also because adopting this design methodology we obtain a well documented and easily maintainable software with a complete traceability of the requirements to the code. This is the result of using: a process that includes models describing all the important aspects of a MAS and a specifically conceived CASE tools that ensures an high level of coherence and continuous checks in the design.

6. ACKNOWLEDGMENTS

We would like to thank A. Luparello, M. Parisi, V. Savarino and S. Sorace for their contribution in realizing the experiment discussed in this paper.

References

[1] A.Chella, S. Gaglio, and R. Pirrone. Conceptual representations of actions for autonomous robots. *Robotics and Autonomous Systems*, 34:251–263, 2001.

[2] F. Bergenti and A. Poggi. Exploiting uml in the design of multi-agent systems. *ESAW Worshop at ECAI*, 2000.

[3] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 1991.

Design (without pattern reuse)	3
Code Production (without pattern reuse)	2
Testing (without pattern reuse)	2
Total amount of work (without pattern reuse)	**7**
Design (with pattern reuse)	2
Code Production (with pattern reuse)	1
Testing (with pattern reuse)	1
Total amount of work (with pattern reuse)	**4**

Table 2. Amount of work (in months) spent in the development of the software system for the surveillance robotic application

[4] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: The tropos project. In *To appear in Information Systems*, Elsevier, Amsterdam, The Netherlands, 2002.

[5] A. Chella, A. Frixione, and S. Gaglio. A cognitive architecture for artifi cial vision. *Artifi cial Intelligence*, 98(1-2):73–111, 1997.

[6] A. Chella, A. Frixione, and S. Gaglio. An architecture for autonomous agents exploiting conceptual representations. *Robotics and Autonomous Systems*, 25:231–240, 1998.

[7] A. Chella, A. Frixione, and S. Gaglio. Understanding dynamic scenes. *Artifi cial Intelligence*, 123:89–132, 2000.

[8] A. Collinot and A. Drogoul. Using the cassiopeia method to design a soccer robot team. *Applied Articial Intelligence (AAI) Journal*, 12(2-3):127–147, 2000.

[9] M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. Introducing pattern reuse in the design of multi-agent systems. In *AITA'02 workshop at NODe02*, Erfurt, Germany, 8-9 October 2002.

[10] S. Cranefi eld and M. Pruvis. Uml as an ontology modelling language. In *Workshop on Intelligent Information Integration*, 1999.

[11] S. A. DeLoach, M. F. Wood, and C. H. Sparkman. Multiagent systems engineering. *International Journal on Software Engineering and Knowledge Engineering*, 11(3):231–258.

[12] Foundation for Intelligent Physical Agents. *FIPA Content Languages Specifi cation*, 2001.

[13] I. Infantino, M. Cossentino, and A. Chella. An agent based multilevel architecture for robotics vision systems. In *The 2002 International Conference on Artifi cial Intelligence*, Las Vegas (NV), USA, June 24-27 2002. ICAI'02.

[14] J.Ulrich and J.Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *IEEE International Conference on Robotics and Automation*, page 1572, Leuven, Belgium, July 15-19.

[15] J.Ulrich and J.Borenstein. Vfh*: Local obstacle avoidance with look-ahead verifi cation. In *IEEE International Conference on Robotics and Automation*, pages 2505–2511, San Francisco (CA), USA, April 2000.

[16] M.Cossentino and C. Potts. A case tool supported methodology for the design of multi-agent systems. Las Vegas (NV), USA, June 24-27 2002. The 2002 International Conference on Software Engineering Research and Practice, SERP'02.

[17] P. O'Brien and R. Nicol. Towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, 1998.

[18] J. Odell, H. V. D. Parunak, and B. Bauer. Extending uml for agents. In *AOIS Workshop at AAAI 2000*, Austin, Texas, July 2000.

[19] M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–315, 2000.