# Agent-Based Modeling of a Mobile Robot to Detect and Follow Humans

José M. Gascueña and Antonio Fernández-Caballero

Universidad de Castilla-La Mancha, Escuela de Ingenieros Industriales de Albacete &
Instituto de Investigación en Informática de Albacete, 02071-Albacete, Spain
{jmanuel,caballer}@dsi.uclm.es

**Abstract.** This paper introduces a multi-agent system (MAS) approach using the detailed process provided by Prometheus methodology for the design of a moving robot application for the detection and following of humans. Our conjecture is that complex autonomous robotic systems have to be fully modeled in their initial design stages by means of agent-based technology. The application has been completely modeled with the Prometheus Design Tool (PDT), which offers full support to Prometheus methodology.

**Keywords:** Multi-agent systems, Agent-based software engineering, Mobile robots, Surveillance.

## 1 Introduction

At present, there are many applications that benefit from the use of mobile robots that incorporate the capability of following persons. Some examples are carrying objects that people working in hospitals, airports, museums, or domestic environments need; or detecting and following intruders. In literature, several studies that use information picked up by devices mounted on a robot to track humans can be found. It is usual to use a camera to detect faces or color blobs, or to follow a contour [13]. Other researchers implement the following task by using information provided by a laser [7]. A hybrid approach is considered in [10], [16], where visual information provided by a camera and information gotten with a laser are used jointly. In addition to the vision sensor, a voice recognition sensor is mounted on the mobile robot in [8] to follow humans in an outdoor environment. Another option involves placing a tracking device on humans [2]. For example two LEDs that are detected by a camera mounted on a robot; or an ultrasonic transponder that allows a robot ultrasonic sensor to distinguish between persons and obstacles.

The works cited previously do not use a methodology that allows requirements capture and design before carrying out the application implementation. Our proposal is to introduce Software Engineering techniques, as these produce a very well documented application from requirements up to implementation [14], [6]. Moreover, using a methodology allows sharing the same terminology, annotation, models, and development processes [1].

A. Håkansson et al. (Eds.): KES-AMSTA 2009, LNAI 5559, pp. 80–89, 2009.

Like humans, robots need a certain level of autonomy, reactivity, pro-activity and social ability to perform their tasks. These characteristics are often cited as a rationale for adopting agent technology [17]; so an agent-oriented methodology will be useful for modeling these kinds of systems. In the last few years a great number of agent-oriented methodologies have been proposed, but only some of them have been applied to develop robotic applications. As far as we know, the only agent-oriented methodologies used to analyze and design a robotic system are Cassiopeia [3], MaSE [5], PASSI [4], and the methodology proposed in [9] that uses concepts from GAIA, Mas-CommonKADS and MaSe methodologies. INGENIAS has been tested in an advanced surveillance system composed of different types of sensors [12].

This paper presents how the detailed process provided by Prometheus methodology has been used to design a robotics application, namely the detection and following of a person, using a multi-agent system (MAS) approach. We have chosen this methodology because it provides a collection of guidelines helping to determine the elements (for instance, agents and interactions) that form the MAS. These guidelines are also helpful to the experts in MAS development. They will be able to transmit their experience to other users through explaining why and how they have obtained the different elements of the agent-based application. In addition, Prometheus is also useful because it explicitly considers agent perceptions and actions as modeling elements. In robotics, percepts are environment data collected by several robot sensors (temperature, light, distance, etc) and actions represent the control carried out by the robot actuators (motors, LEDs, and so on). Lastly, the use of plans also seems a good fit for developing robotic systems.

## 2 Overview of the Prometheus Methodology

Prometheus [11] is defined as proper detailed process to specify, implement and test/debug agent-oriented software systems. It offers a set of detailed guidelines that includes examples and heuristics, which provide a better understanding of what is required in each step of the development. This process incorporates three phases. The system specification phase identifies the basic goals and functionalities of the system, develops the use case scenarios that illustrate its functioning, and specifies which are the inputs (percepts) and outputs (actions). It obtains the analysis overview diagram, scenarios diagram, goal overview diagram, and system roles diagram. The architectural design phase uses the outputs produced in the previous phase to determine the agent types that exist in the system and how they interact. It obtains the data coupling diagram, agent-role diagram, agent acquaintance diagram, and system overview diagram. The detailed design phase focuses on developing the internal structure of each agent and how each agent will perform its tasks within the global system. It obtains agent overview and capability overview diagrams. Finally, Prometheus details how the entities obtained during the design are transformed into the concepts used in a specific agent-oriented programming language (JACK). The design process for Prometheus methodology is supported by Prometheus Design Tool (PDT) [15].

## 3 System Specification

The process to detect and follow moving objects using the robot is described next. The robot is moving randomly around the environment while the images collected are shown to the guard (state *wandering*). After some elapsed time (*Timer_P*) the robot stops in order to analyze the images captured in that instant (state *detecting*). After that, if movement has been detected, (1) information about the detected blob is obtained, and, (2) the guard is warned to decide if the robot should follow the blob or not. The process to follow persons is started (state *following*) if he chooses to follow it (*Follow_P*). When the robot is wandering, the guard may perceive that something is moving in the environment, according to the images displayed on his interface. In that case, the guard orders (*Detect_P*) that the images are analyzed to check if there is or not movement. If the image analysis does not detect movement, then the robot goes on moving randomly. In order to achieve tracking an object correctly (state *following*) the images are captured, displayed, and analyzed continuously in order to obtain blob information. The object is followed until the tracking phase finishes. This condition can be satisfied by three different reasons: (1) the guard has decided not to continue to follow the target (*Follow stop_P*), (2) the target is out of the field of vision, or, (3) it is impossible to follow it because some physical inaccessibility is encountered in the environment (for example, the target takes a staircase). After that, the robot wanders again.

Usually, the System Specification phase begins with the analysis overview diagram, which shows the interactions between the system and the environment (see Fig. 1). An actor is an external entity – human or software/hardware – that interacts with the system. At this level, firstly, an actor for each device mounted on the robot (sonar, bumpers, camera, and wheels) has been identified; there is also a *Guard_A* actor to represent a human that interacts with the system, and a *Timer_A* actor which submits time percepts (*Timer_P*) to the system. There are two scenarios (*Motion detection scenario* and *Object following scenario*) that correspond to the main requirements of the system, and another scenario (*Start system scenario*) to represent the robot components initialization process. Secondly, the information that comes into the system from the environment has been identified (percepts). It corresponds to impacts detected by the bumper device (*Collision_P*), images captured by the camera (*Image_P*), distance to obstacles/targets perceived by the sonar (*Distance_P*), and orders issued by the guard to control the change of the system state (*Detect_P*, *Follow_P*, *Follow stop_P*). On the other hand, everything produced on the actors by the system is also identified (actions). It corresponds to the camera movements carried out based on the tilt, pan and zoom parameters provided (*Set camera focus_a*), commands to control wheel motion (*Set direction_a*, *Stop,_a*, *Move_a*), and an action *Show images_a* to show the images captured. *Show results_a* also highlights with a square the image regions where movement has been detected.

Scenarios are specified in more detail in a scenario diagram. A scenario is a sequence of structured steps – labeled as action (A), percept (P), goal (G), or other scenario (S) – that represents a possible execution way of the system. As an example, *Object following scenario* begins with the order given by the guard in order to follow the blob detected (step 1, P). Then, images are captured (step 2, G) and analyzed

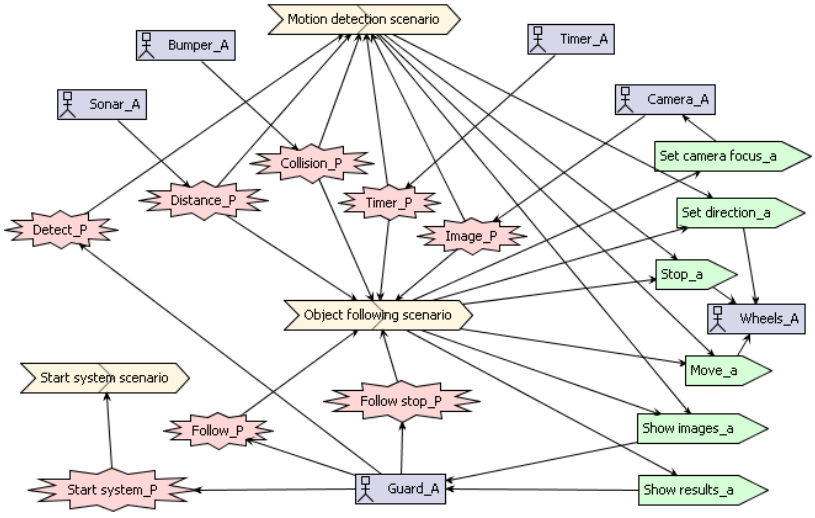


**Fig. 1.** Analysis Overview Diagram

(step 3, G), information about the blob to be followed is gotten (step 4, G), and analysis results are displayed (step 5, A). Based on this information, the robot is oriented to follow the object (step 6, A) and moves towards it (step 7, A). The scenario returns to step 2 until the guard orders to stop object following (step 8, G). The tracking phase is finished when the goal *Finish following* is achieved (step 9, G).

In our MAS approach, several agents communicate and coordinate to pursue the *Environment surveillance* common goal. A goal is associated for every scenario in order to represent the goal that the scenario is intended to achieve. So, in goal overview diagram there are three goals (*Object following*, *Movement detection*, and *Start system*) related to the scenarios identified (*Object following scenario, Motion detection scenario* and *Start system scenario*, respectively). And they contribute to satisfy the common parent goal *Environment surveillance*. Likewise other goals, they are also decomposed into several sub-goals to denote how to achieve each parent goal. Detection and following processes use information provided by the sonar (to avoid obstacles), the bumper (to control collisions), the guard, the information captured by the camera, and perform commands on the wheels (to move robot). So, there are common sub-goals to accomplish *Movement detection* and *Object following* goals.

The roles are identified by clustering goals and linking perceptions and actions (see Fig. 2). *Start System_R* role handles the guard's request to start the robot devices. *Control Collision_R* role is responsible for achieving *Control Collision* goal, for which it needs inputs detected by the physical bumper device. *Observe environment sonar_R* uses the sonar to perceive distances to obstacles in order to avoid them. *Management guard order_R* aims to meet the guard's orders to control the system operation, which has already been started. These orders correspond to perceptions that allow to start/stop the tracking phase (*Follow_P, Stop follow_P*), and to analyze the images (*Detect_P*). *Wander_R* objective is to control the robot "wandering" process. It consists in randomly moving the robot around the environment, avoiding obstacles

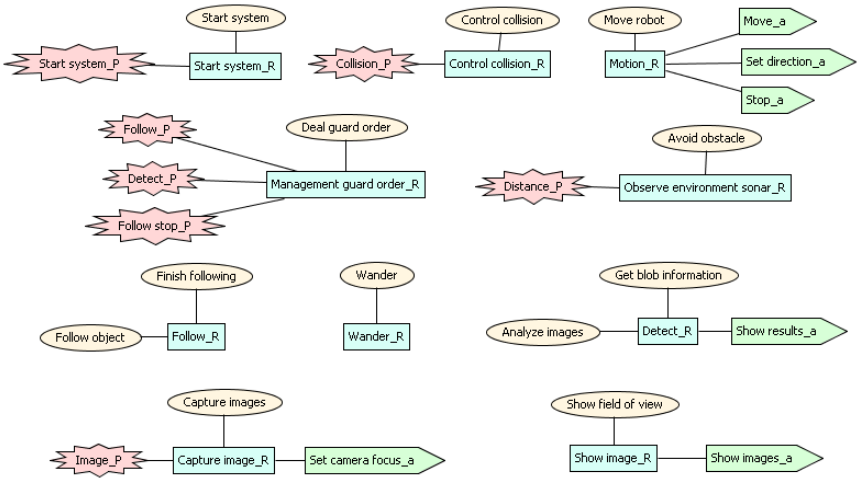


**Fig. 2.** System Roles Diagram

and controlling situations when a collision has been detected. *Follow_R* is responsible for controlling the robot's movement when the system is following an object. *Follow_R*/*Wander_R* roles do not include perceptions from the environment or actions on the environment, but as it will be described later on, it uses information obtained from physical sensors different from the camera, and therefore they need to "communicate" with the roles responsible for achieving *Follow object*/*Wander* sub-goals (*Avoid obstacle*, *Move robot*, *Control collision*). *Detect_R* is responsible for the goals of analyzing images captured by the camera, getting information from the detected moving blob, and performing an action to display results to the guard. *Capture Image_R* perceives images from the environment (*Image_P* percept), and moves the camera to set the camera focus (*Set_focus_a* action) to capture images in an optimum way (*Capture image* goal). *Show Image_R* is responsible for displaying the camera field of view to the guard. To satisfy this goal, *Show Image_a* action is executed when no movement is detected. *Motion_R* uses wheels to move the robot around the area (*Move robot* goal). This is controlled by actions that allow to stop, move and set the motion direction of the robot (*Stop_a*, *Move_a*; *Set motion direction_a*).

It has been shown in previous descriptions that there are entities, such as goals, which appear in several diagrams. This means that updating some diagram may lead to the need of updating another diagram when taking an iterative approach.

## 4 Architectural Design

One task carried out in this phase is to decide the agent types (as collections of roles). This is drawn in the agent-role grouping diagram. In our case we have grouped (1) *Start System_R* and *Management guard order_R* roles into *Central* agent, (2) *Wander_R* and *Follow_R* roles into *Motion Manager* agent, and, (3) *Show image_R* and *Detect_R* roles into *Image Manager* agent. Finally, *Control Collision_R*, *Observe environment sonar_R*, *Motion_R*, *Capture Image_R* roles are related with *Bumper*, *Sonar*, *Wheels*, and

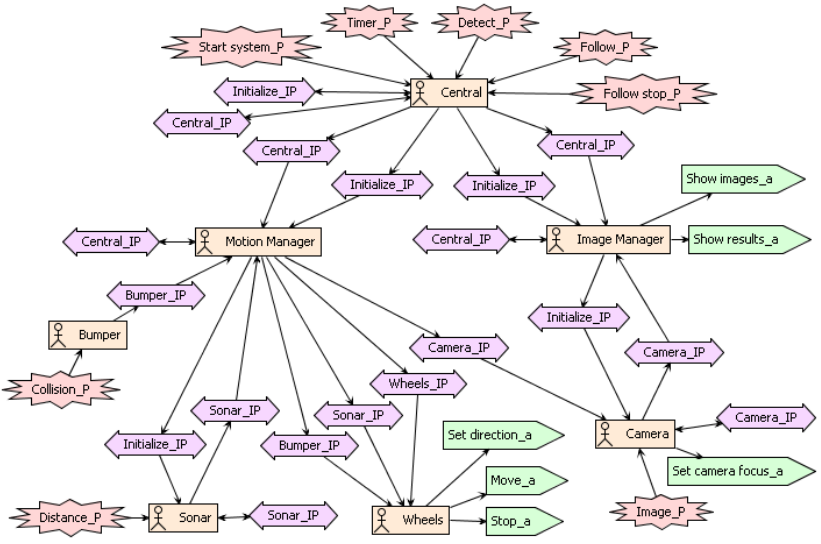


**Fig. 3.** System Overview Diagram

*Camera* agents, respectively. An agent is responsible for the functionalities – roles – related. Once roles have been grouped into agents, information about percepts and actions related to roles, depicted in system roles diagram, it is automatically propagated and linked with the agents in the system overview diagram (see Fig 3).

Once the agents have been identified, the next task is to define agent conversations (interaction protocols - IP) in order to describe what should happen to realize the specified goals and scenarios. Fig. 3 shows the system overview diagram for our system design. *Initialize_IP* means that there are communications between *Central*, *Motion Manager*, *Image Manager*, *Sonar* and *Camera* agents when the system is started for activating the sonar and setting the camera initial parameters. *Bumper_IP* specifies interactions between agents (*Bumper*, *Motion Manager* and *Wheels*), and between agents and environment through *Bumper_A* and *Wheels_A* actors, which occurs when the robot collides with something – the robot should stop and establish a new direction, denoted by actions, in order to continue moving. When the *Bumper* agent perceives that there has been a collision, there is a communication with the *Motion Manager* agent through *Collision_M* message. Then, the *Motion Manager* agent sends messages to the *Wheels* agent to execute the actions mentioned. Collisions occur because the sonar has not been able to detect an obstacle on time. *Sonar_IP* includes messages exchanged between *Sonar*, *Motion Manager* and *Wheels* agents as a result of using information provided by the physical device sonar (it measures the distance from an obstacle to the robot). In this protocol, the *Wheels* agent also executes actions to stop the robot and to orient it towards a new direction when the sonar device detects an obstacle. *Wheels_IP* represents the possible messages sent from the *Motion Manager* agent to the *Wheels* agent in order to execute an action on the robot's wheels. *Central_IP* contains messages sent from the *Central* agent to manager agents (*Motion Manager* and *Image Manager*) to monitor the robot's state (*wandering*,

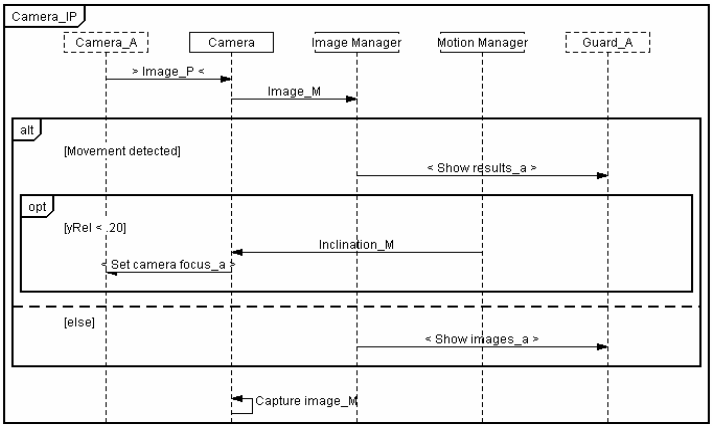


**Fig. 4.** Camara Protocol Diagram

*following*, *detecting*) according to the orders provided by the guard (*Detect_P*, *Follow_P*, *Follow stop_P* percepts) or end of a time slice (*Timer_P* percept). *Wander_M* message is sent to the *Motion Manager*, and it includes 'start_wander', 'continue_wander' or 'stop_wander' value to control the wandering state. The same idea is used with *Follow_M* and *Detect_M* messages sent to the *Motion Manager* and the *Image Manager*, respectively.

Finally, Fig. 4 details the *Camera_IP* interaction protocol internal structure, where interactions involve three agents and two actors (identified by the dotted squares in the diagram). As we can notice, the interaction with the environment is carried out by actors (percepts originated by an actor and going to an agent, whereas actions go from an agent to an actor). Firstly, *Camera_A* actor sends *Image_P* percept, which contains the captured frame to *Camera* agent. This agent sends the information perceived to *Image Manager* agent through *Image_M* message in order to determine if there is motion or not (these options are represented by using an alternative box). If the *Image Manager* evaluates that there is no motion, then it shows the image on the graphical guard interface using *Show images_a* action. Otherwise, it shows an image with a frame on the detected moving blob using *Show results_a* action, and an optional box (opt) will be executed if [yRel <.20] is satisfied. yRel is calculated by *Motion Manager*, only when some object is being followed. This optional box means that the *Motion Manager* agent sends an *Inclination_M* message to the *Camera* agent. Next, the *Camera* agent executes *Set focus_a* action using information about new camera focus contained in the message received. *Camera* agent is continuously receiving images captured by the *A_Camera* actor. This is modeled with *Camera* agent sending to itself an idle *Capture image_M* message, so a new image is captured.

The agent acquaintance diagram contains communication links between agents. It is automatically generated from information messages included in the interaction protocols. In short, there is a hierarchical communication between agents. *Central* sends messages to *Motion Manager* and *Image Manager* depending on the robot's state. The *Motion Manager* sends messages to *Camera* and *Wheels* agents in order to move robot mobile components. Moreover, it receives messages from the *Bumper* and

*Sonar* agents with the information they have collected. *Image Manager* receives messages from *Camera* agent with images perceived in order to detect if there is motion or just to show them.

## 5 Detailed Design

In this phase, the internal details of each agent are specified in a way that is consistent with its related roles and the interface that has been specified with both the environment (percepts and actions) and other agents (messages). This section only shows the Motion Manager agent internal structure (see Fig. 5) as an example. This agent is responsible for handling the movement of the robot's mobile components (camera and wheels). It pursues Wander and Follow object goals related to the roles associated. In order to satisfy these goals it is necessary to achieve Avoid obstacle, Move robot, Control collision sub-goals, which are pursued by Sonar, Wheels, and Bumper agents, respectively. Thus, the Motion Manager agent has a communication with these agents. Start sonar_p plan is triggered (this is denoted with a dashed arrow) by Control sonar_M message sent by Central agent. It sends Activate sonar_M message to Sonar agent in order to start perceiving distance measures. After that, it sends an Analyze sonar_M message to itself, which triggers the Control sonar_p plan. Once the sonar has been activated, Store sonar percept_p plan updates continuously Buffer sonar_D data with information received within the Distance_M message sent by Sonar agent. Control sonar_p plan sends Stop_M message to Wheels agent in order to stop the robot when the obstacle detected by the sonar device is in the robot advance direction. After that, New direction_M message is sent (this contains new robot's direction and velocity) to Wheels agent according to the reading made on the data represented with a cylindrical shape. Wandering_D and Following_D are Boolean data that contain whether the robot is in state wandering and following, respectively. Blob_D data is used to calculate the new direction that the robot should take when it is following an object. Finally, it sends itself an Analyze sonar_M message in order to continuously execute the process that controls the sonar information. Control collision_p plan is triggered by Collision_M message, which is sent by the Bumper agent when a collision has been perceived. To ensure robot's progress, this plan uses an algorithm similar to the one used in Control sonar_p plan.

Moreover, a capability has been created for each role related to this agent. The wandering process is executed in *Wandering_p* plan included within *Wandering_c* capability. It consists in setting a new random direction in a regular time slice. *Wandering_p* is triggered by *Wander_M* message sent by *Central* agent (the message contains 'start wander' or 'stop wander') or *Motion Manager* to continue the wandering process (the message contains 'continue wander'). The messages and data which appear in Fig. 5 related to *Follow_C* capability are propagated automatically towards the capability overview diagram for *Follow_C* depicted in Fig. 5b. *Follow_p* plan is included within *Follow_c* capability. *Follow_p* plan determines the procedure used by the robot to move through the environment when it is following an object. It can be triggered for three different reasons: (1) *Central* agent sends a *Follow_M* message that contains 'start follow' to begin the following process, (2) *Central* sends *Follow_M* with information 'stop follow', which leads to send *Stop_M* and to finish the

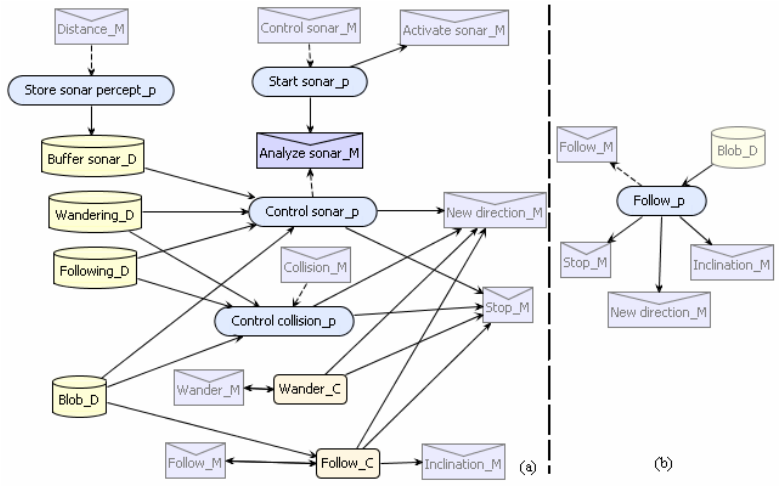


**Fig. 5.** (a) Agent Overview Diagram for *Motion Manager*, (b) Capability Overview Diagram for *Follow_C*

following process, and (3) *Motion Manager* agent sends itself a *Follow_M* message that contains 'continue follow'. Cases one and two use *Blobs_D* data (a) to determine the robot's direction and velocity, which are sent to the *Wheels* agent through *New direction_M* message, and (b) to calculate the camera focus to continue detecting the blob followed, and to send this information within *Inclination_M* message to *Camera* agent. Each plan descriptor includes a procedure field where it is specified in an informal way what the agent will execute.

## 6 Conclusions

A MAS approach using the detailed process provided by Prometheus methodology for the design of a robotic application for the detection and following of humans has been introduced in this paper. Traceability between the entities (concepts) identified along the three phases of the Prometheus methodology has allowed progress in the robotic application design. That is to say, the concepts identified in one phase are helpful in order to identify new concepts that appear in other models of the same phase or another later phase. It has been shown that Prometheus methodology can be used to model the behavior of a single robot that incorporates several sensors. PDT allows automatic code generation from the design, and it can be imported by JACK Development Environment. In the near future, we will try to show the suitability to use it into multi-robot systems.

## Acknowledgments

This work is supported in part by the Spanish Ministerio de Ciencia e Innovación TIN2007-67586-C02 grant, and the Junta de Comunidades de Castilla-La Mancha PBI06-0099 grant.

## References


1. Bordini, R.H., Dastani, M., Winikoff, M.: Current issues in multi-agent systems development. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS(LNAI), vol. 4457, pp. 38–61. Springer, Heidelberg (2007)
2. Chan, H.K., Ye, W., Lam, T.L., Ou, Y., Xu, Y.: Sensor System for a Human-Following Robot. In: International Conference on Automation, Control and Applications, pp. 350–355. Novosibirsk, Russia (2005)
3. Collinot, A., Drogoul, A., Benhamou, P.: Agent Oriented Design of a Soccer Robot Team. In: 2nd International Conference on Multi-Agent Systems (ICMAS 1996), pp. 41–47 (1996)
4. Cossentino, M., Sabatucci, L., Chella, A.: A possible approach to the development of robotic multi-agent systems. In: IEEE/WIC Conference on Intelligent Agent Technology (IAT 2003), pp. 13–17 (2003)
5. DeLoach, S., Matson, E., Li, Y.: Applying Agent Oriented Software Engineering to Cooperative Robotics. In: 15th International Florida Artificial Intelligence Research Society Conference, pp. 391–396 (2002)
6. Gascueña, J.M., Fernández-Caballero, A.: Towards an integrative methodology for developing multi-agent systems. In: 1st International Conference on Agents and Artificial Intelligence, ICAART 2009 (2009)
7. Gockley, R., Forlizzi, J., Simmons, R.: Natural Person Following Behavior for Social Robots. In: ACM/IEEE International Conference on Human-Robot Interaction (HRI 2007), pp. 17–24 (2007)
8. Inamura, T., Shibata, T., Matsumoto, Y., Inaba, M., Inoue, H.: Finding and following a human based on on-line visual feature determination through discourse. In: International Conference on Intelligent Robots and Systems (IROS), pp. 348–353 (1998)
9. Jiménez Builes, J.A., Vallejo Valencia, M., Ochoa Gómez, J.F.: Methodology for the Analysis and Design of Multi-Agent Robotic Systems: MAD-Smart. Revista Avances en Sistemas e Informática 4(2), 61–70 (2007)
10. Kobilarov, M., Hyams, J., Batavia, P., Sukhatme, G.S.: People tracking and following with mobile robot using an omnidirectional camera and a laser. In: IEEE International Conference on Robotics and Automation, pp. 557–562 (2006)
11. Padgham, L., Winikoff, M.: Developing intelligent agents systems: A practical guide. John Wiley and Sons, Chichester (2004)
12. Pavón, J., Gómez-Sanz, J.J., Fernández-Caballero, A., Valencia-Jiménez, J.J.: Development of intelligent multi-sensor surveillance systems with agents. Robotics and Autonomous Systems 55(12), 892–903 (2007)
13. Schlegel, C., Illmann, J., Jaberg, K., Schuster, M., Wörz, R.: Vision based person tracking with a mobile robot. In: 9th British Machine Vision Conference (BMVC), pp. 418–427 (1998)
14. Sokolova, M.V., Fernández-Caballero, A.: Facilitating MAS complete life cycle through the Protégé-Prometheus approach. In: Nguyen, N.T., Jo, G.S., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2008. LNCS, vol. 4953, pp. 63–72. Springer, Heidelberg (2008)
15. Thangarajah, J., Padgham, L., Winikoff, M.: Prometheus Design Tool. In: 4th International Conference on Autonomous Agents and Multi-Agent Systems, pp. 127–128 (2005)
16. Zivkovic, Z., Kröse, B.: People Detection Using Multiple Sensors on a Mobile Robot. In: Unifying Perspectives in Computational and Robot Vision. Lecture Notes in Electrical Engineering, pp. 25–39 (2008)
17. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review 10(2), 115–152 (1995)