Agent-oriented modeling and development of a person-following mobile robot

José Manuel Gascueña^a, Antonio Fernández-Caballero^{a,b,*}

^a Instituto de Investigación en Informática de Albacete (I3A), Universidad de Castilla-La Mancha, 02071 Albacete, Spain

^b Departamento de Sistemas Informáticos, Escuela de Ingenieros Industriales de Albacete, Universidad de Castilla-La Mancha, 02071 Albacete, Spain

ARTICLE INFO

Keywords:

Multi-agent systems
Agent-oriented Software Engineering
Mobile robots
Surveillance

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. Code has been generated in the agent-oriented programming language JACK.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

In the last few decades, the field of surveillance systems has captured the attention of the industry and investigation in the academic realm (Fernández-Caballero, Gómez, & López-López, 2008; López, Fernández-Caballero, Fernández, Mira, & Delgado, 2006). It is usual to find surveillance systems made up of heterogeneous devices which make it possible to recover different flow of information (video, audio, images, scalar data) from the scenario to carry out an analysis and interpretation of the scene in real time (Cucchiara, 2005; Valera & Velastin, 2005). From the point of view of image processing, these systems are based on distributing the processing capabilities and using embedded signal processing devices to gain from the advantages of the scalability and robustness of the distributed systems. The main problems which need to be solved in these systems are: integrating data obtained from different sensors, establishing a correspondence of the signals in time and space and coordinating and distributing the processing task and video communication.

Surveillance systems are fundamentally used for three reasons. First, to guarantee public safety in public spaces (e.g. train stations, subways or airports) and/or private spaces, detecting and preventing possible vandal or criminal attacks; second, in order to carry out a forensic analysis of the incidents occurred and third, to obtain statistics about people, vehicles and their behavior. In particular, mobile robots could be equipped with different sensors in order to perform surveillance tasks because they are able to obtain a vision of the objects of interest from a different perspective, and to

accede to zones that are inaccessible to fixed cameras or that are dangerous for the humans. For this reason, the main goal of this article is to design a moving robot application for the detection and following of humans. The potential of the practical use of the concepts and technologies related with the Prometheus agent-oriented Software Engineering methodology (Padgham & Winikoff, 2004) is evaluated to satisfy this goal.

The rest of the article is organized as follows. First, it is introduced why agents are a good choice for solving the problems which appear and are dealt with in surveillance systems. Next, a collection of works which use robots in related subjects to surveillance systems is summed up. Moreover, it is emphasized the use of agent-based technology in surveillance-related robotic applications and why Prometheus has been selected. Afterwards, the agent-oriented modeling of our system from requirement capture to code generation through the phases proposed in Prometheus is presented. Finally, some conclusions are offered.

2. Why agents are relevant in surveillance systems

There is no universally accepted definition for the term “agent”, but there is a wide range of perspectives in function of the application domain, the author and so on. Franklin and Graesser (1996) state: “An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future”. Any agent, in accordance with this definition, satisfies the four properties as indicated next (Wooldridge & Jennings, 1995): (1) autonomy – agents operate without the direct intervention of humans or other agents and have some kind of control over their actions and internal state; (2) social ability – agents interact with other agents (and possibly humans) via some kind of agent-communication language; agents collaborate for the sake of

* Corresponding author at: Instituto de Investigación en Informática de Albacete (I3A), Universidad de Castilla-La Mancha, 02071 Albacete, Spain. Tel.: +34 967 599200; fax: +34 967 599224.

E-mail addresses: jmanuel@dsi.uclm.es (J.M. Gascueña), caballer@dsi.uclm.es (A. Fernández-Caballero).

performing tasks; (3) reactivity – agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined) and respond in a timely fashion to changes what occurs in it; in order to respond effectively to changes, agents have to know at each instant their surrounding “world”; (4) pro-activeness – agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

The capability to communicate makes it possible for agents to work together to solve complex problems which cannot be dealt with by a single agent, this being the essence of multi-agent systems (MAS) (Huhns & Stephens, 1999). MAS are noted for the fact that they are made up of collections of potentially independent and autonomous agents, usually heterogeneous, which work together to solve a problem which goes beyond their individual capabilities. MAS are appropriate within domains in which the necessary knowledge to solve a problem is distributed along different places. The solution to the problem depends on the coordination of the tasks to be carried out by different entities with different capabilities, usually without the supervision of a single centralized coordinator. For example, defense applications are performed in highly decentralized and heterogeneous environments and/or require the incorporation of intelligent decision making. These characteristics make the technologies, techniques and algorithms used within the scope of MAS adequate to be applied in military domain applications (Pechoucek, Thompson, & Voos, 2008) such as logistics, manned and unmanned air traffic control, simulation and training. Likewise, on the one hand, Patricio et al. (2008) highlight the suitability of using a MAS for video surveillance because (1) the loose coupling nature of a multi-agent architecture allows more flexibility in the communication process and (2) the ability to assign responsibilities to each agent is ideal to solve complex tasks in a surveillance system. These complex tasks entail the use of coordination and cooperation mechanisms and dynamic configuration, which are widely used in the MAS community (d’Inverno et al., 1997). On the other hand, intelligence distribution in MAS allows dealing with questions that turn up in the development of surveillance systems (bandwidth, productivity, speed, robustness, autonomy, scalability) (Pavón, Gómez-Sanz, Fernández-Caballero, & Valencia-Jiménez, 2007). Tran (2006) has recently introduced an excellent comparison among the agent-oriented methodologies cited previously.

In summary, the basic characteristics of the agents (autonomy, reactivity, pro-activeness and social ability), along with the characteristics of MAS (distributed data management, low coupling, robustness, communication and coordination between autonomous entities), suggest that they are good choices to solve the problems dealt with in surveillance systems.

3. Use of agent-based technology in surveillance-related robotic applications

At present, there are many applications that benefit from the use of mobile robots that incorporate the capability of following persons (e.g. Braun, Szentpetery, & Berns, 2005; Chen & Birchfield, 2007; Feyrer & Zell, 1999; Gigliotta, Caretti, Shokur, & Nolfi, 2005; Ohya, Nagumo, & Takahata, 2002; Yoshimi et al., 2006). 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 (Schlegel, Illmann, Jaberg, Schuster, & Würz, 1998). Other researchers implement the following task by using

information provided by a laser (Gockley, Forlizzi, & Simmons, 2007). A hybrid approach is considered in Kobilarov, Hyams, Batavia, and Sukhatme (2006) and Zivkovic and Kröse (2008), 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 Inamura, Shibata, Matsumoto, Inaba, and Inoue (1998) to follow humans in an outdoor environment. Another option involves placing a tracking device on humans (Chan, Ye, Lam, Ou, & Xu, 2005). 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 (Gascueña & Fernández-Caballero, 2010). Indeed, modeling the motion detection task (López, Fernández-Caballero, Mira, Delgado, & Fernández, 2006; Mira, Delgado, Fernández-Caballero, & Fernández, 2004) and related applications (Fernández-Caballero, Gómez, et al., 2008; Fernández-Caballero et al., 2007; Fernández-Caballero et al., 2008) has been the main concern of recent works. Our proposal is to introduce Knowledge Engineering and Software Engineering techniques, as these produce a very well documented application from requirements up to implementation (Gascueña & Fernández-Caballero, 2007, 2009a, 2009b; Sokolova & Fernández-Caballero, 2007, 2008). Moreover, using a methodology allows sharing the same terminology, annotation, models, and development processes (Bordini, Dastani, & Winikoff, 2007).

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 (Wooldridge & Jennings, 1995); 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 (Collinot, Drogoul, & Benhamou, 1996), MaSE (DeLoach, Matson, & Li, 2002), PASSI (Cossentino, Sabatucci, & Chella, 2003), and the methodology proposed in Jiménez, Vallejo, and Ochoa (2007) 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 (Pavón et al., 2007).

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.

4. Overview of the Prometheus methodology

Prometheus (Padgham & Winikoff, 2004) is defined as proper detailed process to specify, implement and test/debug

Fig. 1. Overview of the Prometheus methodology.

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 (see Fig. 1).

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) (Thangarajah, Padgham, & Winikoff, 2005).

5. System specification

The process to detect and follow moving objects using the robot is depicted in Fig. 2. 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

Fig. 2. The robot's states.

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. 3). 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 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.

Fig. 3. Analysis overview diagram.

Edit Scenario - Object following scenario					
	Type	Name	Role	Description	Data
1	P	Follow_P	Management guard order_R	Order given by guard to follow a blob	
2	G	Capture images	Capture image_R	It captures environment images	Buffer images_D
3	G	Analyze images	Detect_R	It analyzes images captured	Buffer images_D
4	G	Get blob information	Detect_R	It gets blob information in order to follow it	Blob_D
5	A	Show results_a	Show image_R	It shows detection results	Buffer images_D, Blob_D
6	A	Set direction_a	Motion_R	It sets the direction in order to follow the blob	Blob_D
7	A	Move_a	Motion_R	It moves wheels to follow the object detected	
8	G	Deal guard order	Management guard order_R	Deal guard orders, if there is	
9	G	Finish following	Follow_R	It finishes the following	

A -> Action G -> Goal O -> Others P -> Percept S -> Scenario

Fig. 4. Object following scenario.

Fig. 5. Goal overview diagram.

Fig. 6. System roles diagram.

Scenarios are specified in more detail in a scenario diagram. A scenario is a sequence of structured steps – labeled as action, percept, goal, or other scenario – that represents a possible execution way of the system. As an example, Fig. 4 illustrates the process performed by the system to follow the moving object detected (*Object following scenario*). This scenario begins with the order given by the guard in order to follow the blob detected (step 1). Then, images are captured (step 2) and analyzed (step 3), information about the blob to be followed is gotten (step 4), and analysis results are displayed (step 5). Based on this information, the robot is oriented to follow the object (step 6) and moves towards it (step 7). The scenario returns to step 2 until the guard orders to stop object following (step 8). The tracking phase is finished when the goal *Finish following* is achieved (step 9).

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 (see Fig. 5) 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. 6). In Fig. 6, it is necessary to highlight that *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*). On the other hand, wandering process consists in randomly moving the robot around the environment, avoiding obstacles and controlling situations when a collision has been detected. Table 1 summarizes the description of the roles identified.

6. 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 (see Fig. 7). 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)

Table 1
Roles description.

Role	Description
<i>Start System_R</i>	It handles the guard's request to start the robot devices
<i>Control Collision_R</i>	It is responsible for achieving <i>Control Collision</i> goal, for which it needs inputs detected by the physical bumper device
<i>Observe environment sonar_R</i>	It uses the sonar to perceive distances to obstacles in order to avoid them
<i>Management guard order_R</i>	It 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 (<i>Follow_P</i> , <i>Stop follow_P</i>), and to analyze the images (<i>Detect_P</i>)
<i>Wander_R</i>	Its objective is to control the robot "wandering" process
<i>Follow_R</i>	It is responsible for controlling the robot's movement when the system is following an object
<i>Detect_R</i>	It 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
<i>Capture Image_R</i>	It perceives images from the environment (<i>Image_P</i> percept), and moves the camera to set the camera focus (<i>Set_focus_a</i> action) to capture images in an optimum way (<i>Capture images</i> goal)
<i>Show image_R</i>	It is responsible for displaying the camera field of view to the guard. To satisfy this goal, <i>Show images_a</i> action is executed when no movement is detected
<i>Motion_R</i>	It uses wheels to move the robot around the area (<i>Move robot</i> goal). This is controlled by actions that allow to stop, move and set the motion direction of the robot (<i>Stop_a</i> , <i>Move_a</i> , <i>Set direction_a</i>)

Fig. 7. Agent-role grouping diagram.

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 *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. 8).

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. 8 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*, *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' values to control the wandering state. The same idea is used with *Follow_P* and *Detect_M* messages sent to the *Motion Manager* and the *Image Manager*, respectively.

Finally, Fig. 9 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

Fig. 8. System overview diagram.

Fig. 9. Camera protocol diagram.

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 *Camera_A* 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 (see Fig. 10). *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.

Fig. 10. Agent acquaintance diagram.

7. 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. 11a) 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. 11a related to *Follow_c* capability are propagated automatically towards the capability overview diagram for *Follow_C* depicted in Fig. 11b. *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 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.

It has been shown in the previous figures 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. The Goal overview diagram allows creating AND and OR relations between goals. The Data coupling diagram allows establishing relations between data and roles. On the other hand, Table 2 summarizes the rest of diagrams used in Prometheus and entities that appear in each diagram. Moreover, it is necessary to highlight: (a) *Protocol* entity

Fig. 11. (a) Agent overview diagram for motion manager and (b) capability overview diagram for *Follow_C*.

Table 2
Relation between entities and diagrams used in Prometheus.

Entity	Diagrams						
	Analysis overview	Scenario	System roles	Agent-role grouping	System overview	Agent overview	Capability overview
Action	✓						
Actor	✓						
Agent				✓			
Capability						✓	✓
Data		✓			✓	✓	✓
Goal		✓	✓				
Message					✓	✓	✓
Percept	✓	✓	✓		✓	✓	✓
Plan						✓	✓
Protocol					✓		
Role		✓	✓	✓			
Scenario	✓	✓					

can contain *Agent*, *Actor*, *Message*, *Percept*, and *Action* entities too, and (b) entities related with the Scenario diagram may appear as information associated with a step in some scenario.

8. Code generation

Sometimes the implementation is completely manual starting from the design. This creates the possibility for the design and implementation to diverge. And, this tends to make the design less useful for further work in maintenance and comprehension of the system (Bordini et al., 2007). In that case there is a gap between the design models of the methodologies and the existing implementation languages. To bridge the gap, a methodology should either introduce refined design models that can be directly implemented in an available programming language, or use a dedicated agent-

oriented programming language that provides constructs to implement the high-level design concepts (Dastani, Hulstijn, Dignum, & Meyer, 2004). Prometheus methodology follows the first approach, that is to say, its last phase (the Detailed Design phase) offers models close enough to the concepts used in a specific agent-oriented programming language named JACK (Winikoff, 2005). Hence, the entities obtained during the design are able to be directly transformed into the concepts used in JACK. Table 3 shows which Prometheus entities are translated into their equivalent JACK concepts. Note that some entities (*Actor*, *Goal*, *Protocol*, *Role*, *Scenario*) are not transformed into JACK concepts. The *Action* concept is not transformed into JACK specific concept, but it can be implemented in the associated agent as a method.

The model developed using PDT tool is stored in a “filename.pd” file, where filename is the name provided for the modeling. The code generation process is depicted in Fig. 12 and is summarized as follows. Firstly, the developer uses the option *Generate Code* provided by PDT, which is available in the *Tools* menu, in order to show the ‘code generation dialog’. In this dialog, he/she chooses the code generation directory (in our case JACK) and presses the *Generate* button to generate a JACK skeleton code from the design. This action generates automatically a JACK folder, which includes several subfolders (agents, capabilities, data, events, plans). The agent sub-folder contains a file “.agent” for each agent entity included into the model (the filename is the name provided in the Prometheus model for the agent). The same occurs for the capability, data, message and plan entities created in the model, with the exception of file extension and folder where are stored as depicted in Fig. 12 on the top right.

Table 3
Mapping Prometheus modeling concepts into JACK concepts.

Prometheus entity	JACK concept
Agent	Agent
Capability	Capability
Percept	Event
Message	Event
Plan	Plan
Data	BeliefSet
Action	–

Fig. 12. Code generation process.

Secondly, the developer uses JACK Development Environment (JDE) tool to import the code generated by PDT. To do that, (1) the 'Compiler Utility' submenu available in 'Tools' menu is chosen in JDE; (2) 'Convert Non-JDE JACK' tab is selected for converting existing JACK code generated by PDT into its own JDE project; (3) the folder that contains the code generated by PDT is introduced into content list; (4) the path and name for the JDE project file is specified in order to know where it will be stored; and, finally, (5) when the 'Generate' button is pressed the new JDE project (.prj file) specified previously will be obtained. Also the directory structure shown in Fig. 12 at the bottom is gotten. After carrying out the import, it can be observed that the results obtained are very similar to those ones obtained when PDT is used to generate code from the Prometheus model. Now, the internal structure for the files and their extensions are different in order to be readable by JDE. Finally, after creating some java classes (e.g., classes necessary to create graphical user interfaces, main class to create instances of agents, or other utilities) and completing the generated 'gcode', the JACK program can easily be transferred into Java using the facilities provided by JDE, and executed.

Notice that the developer needs to manually complete in JDE the code that has been imported. For instance, Fig. 13 shows the JACK file that is obtained in the new JDE project after importing the files generated by PDT, associated with PDT *Follow_M* message entity. It is necessary to give a more descriptive name to event's posting method (methodName is always obtained by default). Moreover, the variables associated with the information specified with PDT into the carried information field of message form must be declared. The final code for *Follow_M* event is shown in Fig. 14.

Fig. 13. JACK file generated for *Follow_M* event.

Fig. 14. JACK file generated for *Follow_M* event.

Fig. 15. JACK file generated for *Follow_p* plan.

In this case posting method's name is *Follow_M* and the option variable will contain 'start follow', 'stop follow' or 'continue follow' when *Follow_M* event is sent, such as was described in Section 7. Fig. 15 shows the code obtained for *Follow_p* plan. In this case it is necessary complete the body reasoning method.

9. 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 collaborative systems.

Acknowledgments

This work is supported in part by the Spanish Ministerio de Ciencia e Innovación TIN2007-67586-C02-02 and TIN2010-20845-C03-01 grants, and the Junta de Comunidades de Castilla-La Mancha PII2I09-0069-0994, PII2I09-0071-3947 and PEII09-0054-9581 grants.

References

- Bordini, R. H., Dastani, M., & Winikoff, M. (2007). Current issues in multi-agent systems development. In *Seventh annual international workshop on engineering societies in the agents world. Lecture notes in artificial intelligence* (Vol. 4457, pp. 38–61).

- Braun, T., Szepietery, K., & Berns, K. (2005). Detecting and following humans with a mobile robot. In *EOS conference on industrial imaging and machine vision*.
- Chan, H. K., Ye, W., Lam, T.L., Ou, Y., & Xu, Y. (2005). Sensor system for a human-following robot. In *International conference on automation, control and applications* (pp. 350–355).
- Chen, Z., & Birchfield, S. T. (2007). Person following with a mobile robot using binocular feature-based tracking. In *IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 815–820).
- Collinot, A., Drogoul, A., & Benhamou, P. (1996). Agent oriented design of a soccer robot team. In *Second international conference on multi-agent systems, ICMAS'96* (pp. 41–47).
- Cossentino, M., Sabatucci, L., & Chella, A. (2003). A possible approach to the development of robotic multi-agent systems. In *IEEE/WIC conference on intelligent agent technology, IAT'03* (pp. 13–17).
- Cucchiar, R. (2005). Multimedia surveillance systems. In *The 3rd ACM international workshop on video surveillance and sensor networks* (pp. 3–10).
- Dastani, M., Hulstijn, J., Dignum, F., & Meyer, J. (2004). Issues in multiagent system development. In *Third international joint conference on autonomous agents and multiagent systems (AAMAS'04)* (pp. 922–929).
- DeLoach, S., Matson, E., & Li, Y. (2002). Applying agent oriented software engineering to cooperative robotics. In *Fifteenth international Florida artificial intelligence research society conference* (pp. 391–396).
- d'Inverno, M., Fisher, M., Lomuscio, A., Luck, M., de Rijke, M., Ryan, M., et al. (1997). Formalisms for multi-agent systems. *The Knowledge Engineering Review*, 12, 315–321.
- Feyrer, S., & Zell, A. (1999). Tracking and pursuing persons with a mobile robot. In *International workshop on recognition, analysis, and tracking of faces and gestures in real-time systems* (pp. 83–88).
- Fernández-Caballero, A., Gómez, F. J., & López-López, J. (2008). Road-traffic monitoring by knowledge-driven static and dynamic image analysis. *Expert Systems with Applications*, 35(3), 701–719.
- Fernández-Caballero, A., López, M. T., Mira, J., Delgado, A. E., López-Valles, J. M., & Fernández, M. A. (2007). Modelling the stereovision-correspondence-analysis task by lateral inhibition in accumulative computation problem-solving method. *Expert Systems with Applications*, 33(4), 955–967.
- Fernández-Caballero, A., López, M. T., & Saiz-Valverde, S. (2008). Dynamic stereoscopic selective visual attention (DSSVA): Integrating motion and shape with depth in video segmentation. *Expert Systems with Applications*, 34(2), 1394–1402.
- Franklin, S., & Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent agents III, agent theories, architectures and languages. Lecture notes in computer science* (Vol. 1193, pp. 21–35).
- Gascueña, J. M., & Fernández-Caballero, A. (2007). The INGENIAS methodology for advanced surveillance systems modelling. In *Second international work-conference on the interplay between natural and artificial computation. Lecture notes in computer science* (Vol. 4528, pp. 5410–550).
- Gascueña, J. M., & Fernández-Caballero, A. (2009a). Prometheus and INGENIAS agent methodologies: A complementary approach. *Agent-oriented software engineering IX. Lecture notes in computer science* (Vol. 5386, pp. 131–144).
- Gascueña, J. M., & Fernández-Caballero, A. (2009b). Towards an integrative methodology for developing multi-agent systems. In *First international conference on agents and artificial intelligence, ICAART 2009* (pp. 392–399).
- Gascueña, J. M., & Fernández-Caballero, A. (2010). On the use of agent technology in intelligent, multi-sensory and distributed surveillance. *The Knowledge Engineering Review*, in press.
- Gigliotta, O., Caretti, M., Shokur, S., & Nolfi, S. (2005). Toward a person-follower robot. In *The 2nd robocare workshop*.
- Gockley, R., Forlizzi, J., & Simmons, R. (2007). Natural person following behavior for social robots. In *ACM/IEEE international conference on human-robot interaction, HRI'07* (pp. 17–24).
- Huhns, M. N., & Stephens, L. M. (1999). Multiagent systems and societies of agents. In G. Weiss (Ed.), *Multiagent systems*. The MIT Press.
- Inamura, T., Shibata, T., Matsumoto, Y., Inaba, M., & Inoue, H. (1998). Finding and following a human based on on-line visual feature determination through discourse. In *International conference on intelligent robots and systems, IROS'98* (pp. 348–353).
- Jiménez, J. A., Vallejo, M., & Ochoa, J. F. (2007). Methodology for the analysis and design of multi-agent robotic systems: MAD-Smart. *Revista Avances en Sistemas e Informática*, 4(2), 61–70.
- Kobilarov, M., Hyams, J., Batavia, P., & Sukhatme, G. S. (2006). 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).
- López, M. T., Fernández-Caballero, A., Fernández, M. A., Mira, J., & Delgado, A. E. (2006). Visual surveillance by dynamic visual attention method. *Pattern Recognition*, 39(11), 2194–2211.
- López, M. T., Fernández-Caballero, A., Mira, J., Delgado, A. E., & Fernández, M. A. (2006). Algorithmic lateral inhibition method in dynamic and selective visual attention task: Application to moving objects detection and labelling. *Expert Systems with Applications*, 31(3), 570–594.
- Mira, J., Delgado, A. E., Fernández-Caballero, A., & Fernández, M. A. (2004). Knowledge modelling for the motion detection task: the algorithmic lateral inhibition method. *Expert Systems with Applications*, 27(2), 169–185.
- Ohya, A., Nagumo, Y., & Takahata, M. (2002). Intelligent escort robot moving together with human-human following behavior. In *Twelfth international symposium on measurement and control in robotics*.
- Padgham, L., & Winikoff, M. (2004). *Developing intelligent agents systems: A practical guide*. John Wiley and Sons.
- Patricio, M. A., Castanedo, F., Berlanga, A., Pérez, O., García, J., & Molina, J. M. (2008). Computational intelligence in visual sensor networks: Improving video processing systems. *Studies in Computational Intelligence*, 96, 351–377.
- Pavón, J., Gómez-Sanz, J. J., Fernández-Caballero, A., & Valencia-Jiménez, J. J. (2007). Development of intelligent multi-sensor surveillance systems with agents. *Robotics and Autonomous Systems*, 55(12), 892–903.
- Pechoucek, M., Thompson, S. G., & Voos, H. (2008). *Defense industry applications of autonomous agents and multiagent systems. Whitestein series in software agent technologies and autonomic computing*. A Birkhäuser Basel book. ISBN 978-3-7643-8570-5.
- Schlegel, C., Illmann, J., Jaber, K., Schuster, M., & Wörz, R. (1998). Vision based person tracking with a mobile robot. In *Ninth British machine vision conference, BMVC'98* (pp. 418–427).
- Sokolova, M. V., & Fernández-Caballero, A. (2007). A meta-ontological framework for multi-agent systems design. In *Second international work-conference on the interplay between natural and artificial computation, IWINAC 2007. Lecture notes in computer science* (Vol. 4528, pp. 521–530).
- Sokolova, M. V., & Fernández-Caballero, A. (2008). Facilitating MAS complete life cycle through the Protégé–Prometheus approach. In *The 2nd KES international symposium on agent and multi-agent systems: Technologies and applications, KES-AMSTA 2008* (pp. 63–72).
- Thangarajah, J., Padgham, L., & Winikoff, M. (2005). Prometheus design tool. In *Fourth international conference on autonomous agents and multi-agent systems* (pp. 127–128).
- Tran, Q. N. (2006). *MOBMAS – A methodology for ontology-based multi-agent systems development*. PhD dissertation, School of Information Systems, Technology and Management, University of New South Wales.
- Valera, M., & Velastin, S. A. (2005). Intelligent distributed surveillance systems: A review. *IEEE Proceedings on Vision, Image and Signal Processing*, 152(2), 192–204.
- Winikoff, M. (2005). Jack™ intelligent agents: An industrial strength platform. In *Multi-agent programming languages, platforms applications* (pp. 175–193). Springer. ISBN 0387245685.
- Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2), 115–152.
- Yoshimi, T., Nishiyama, M., Sonoura, T., Nakamoto, H., Tokura, S., Sato, H., et al. (2006). Development of a person following robot with vision based target detection. In *IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 5286–5291).
- Zivkovic, Z., & Kröse, B. (2008). People detection using multiple sensors on a mobile robot. In *Unifying perspectives in computational and robot vision* (pp. 25–39). Springer.