Developing Multi-Agent Systems through Integrating Prometheus, INGENIAS and ICARO-T

Antonio Fernández-Caballero and José M. Gascueña

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

Abstract. A great number of methodologies to develop MAS systems have been proposed in the last few years. But, a perfect methodology that satisfies all the developer necessities cannot be found. This is the reason why different methodologies are studied to create a new one. In this article, a methodology that includes all steps from the capture of requirements to the implementation and deployment of an agent-based application is proposed. In first place, an Analysis Overview Diagram is created to obtain an initial sketch of the application. Afterwards, the model obtained - by following the two first stages proposed by Prometheus methodology – could be integrated into INGENIAS through UML-AT language. Next, the modeling goes on with INGENIAS. Finally, code is generated for the ICARO-T platform.

Keywords: Multi-agent systems, Agent development methodologies.

1 Introduction

Multi-agent systems (MAS) technology is adequate for developing open, complex, and distributed systems, and they offer a natural way of operating with legacy systems [19]. A great number of methodologies to develop MAS systems have been proposed in the last few years. Gaia, Tropos, MaSE, MESSAGE, Prometheus, and INGENIAS are just a few examples. Nonetheless, a unique methodology cannot be general enough to be useful for everyone without some level of customization [4]. Usually, techniques and tools proposed in different methodologies to provide a solution to the specific problem that is being approached are combined. The result is a new methodology fruit of combining several proposals of the analyzed methodologies. In fact, in the literature, methodologies can be found that are influenced by other methodologies that already were proposed previously. For instance, fragments of PASSI, Gaia, and Tropos are considered to define the MAR&A methodology [3].

In this article, INGENIAS is chosen as the basis, due to its recent direction towards model-driven development (MDD) [24] in order to define a new methodology to develop MAS. But, the two first stages proposed in Prometheus [21], namely *system specification* and *architectural design*, are previously integrated in order to solve some current deficiencies in INGENIAS (see section 2). The language used by Prometheus is different from the INGENIAS language. Therefore, in order to use INGENIAS, it is necessary to transform the model obtained with Prometheus into an equivalent INGENIAS model. This transformation can be performed with language UML-AT [7], [8]. Later we propose to continue modeling with INGENIAS. Finally, code is generated for the ICARO-T platform [11]. The result of using the mentioned technologies, Prometheus, INGENIAS, UML-AT and ICARO-T, turns into a new methodology to develop MAS. The process followed in the methodology assists the MAS developer from the capture of requirements to the implementation and deployment of the application.

The article structure is as follows. Section 2 describes the contributions made by INGENIAS and the deficiencies that it presents. Methodologies Prometheus and INGENIAS, as well as the tools that support them, are compared in section 3. In section 4 the phases of the integrative methodology to develop MAS are proposed and described. Finally, some conclusions are offered.

2 Why Start with INGENIAS?

In the initial INGENIAS proposal [23] there are several contributions to develop MAS. First, it offers a meta-model to specify MAS. A MAS is considered from five complementary viewpoints: organization, agent, goals and tasks, interaction, and environment. Second, it adopts the unified software development process (USDP) [17] as a guideline to define the steps necessary to develop the elements and diagrams of MAS during the analysis and design phases. Third, INGENIAS Development Kit (IDK) is a tool that supports the methodology. IDK has integrated a set of utilities that allow model edition, verification, validation, and automatically generate code and documentation.

Now, INGENIAS is being reformulated in terms of the MDD paradigm [24]. Nowadays the use of model-driven engineering (MDE) techniques along the life cycle of software development is gaining more and more interest [28]. The key idea underlying this paradigm is that if the development is guided by models there will be important benefits in fundamental aspects such as productivity, portability, interoperability and maintenance. Therefore, in the MAS field, it seems quite useful to use a methodology such as INGENIAS, which supports this approach. There are some other works using MDE in the area of MAS [27], [18], [16], [10], among others.

Indeed, there are other reasons for studying the methodology INGENIAS and the tools created around. The INGENIAS engineer, connoisseur of the INGENIAS meta-model, can (a) define the meta-model for the domain of a concrete application, (b) personalize the IDK for a specific application domain, and, (c) create transformations to generate source code for the final platform on which the agents will run. There exist some previous experiences to adapt the INGENIAS language to more specific systems. For example, the IDK framework has been used to construct an editor for Holonic Manufacturing Systems. Also the INGENIAS language has been adapted for social simulation environments.

Unfortunately, in our opinion, the process followed in INGENIAS during the analysis and design phases of MAS is very complex and difficult to understand, because it is not clear how the different models are being constructed along the phases, despite the documented general guidelines. Moreover, INGENIAS does not provide any mechanism to discover which will be the agents of the system and their interactions. Thus, it is necessary to raise a process of alternative development that makes system development simpler. In order to make the MAS methodology easy to use for non expert people in the development of such systems, it is necessary that it offers a collection of detailed guidelines, including examples and heuristics, which help better understanding what is required in each step of the development process used in the methodology. These guidelines also serve as a help to the experts in MAS development. They will be able to transmit their experience to other users explaining why and how they have obtained the different elements (agents, interactions, etc.) of the agent-based application.

3 Comparison of Prometheus and INGENIAS

INGENIAS has several advantages as opposed to Prometheus (see Table 1): (a) it follows an MDD approach, (b) it facilitates a general process to transform the models generated during the design phase into executable code. The advantages of Prometheus can be used (following the process to discover which be the agents of the system and its interactions) to enhance INGENIAS. In Table 2 the Prometheus Design Tool (PDT) [22] and INGENIAS Development Kit (IDK) [15] tools are compared. It may be observed that PDT only has one advantage with respect to IDK: it has a mechanism to prioritize parts of a project. In the rest of considered characteristics, IDK equals or surpasses PDT. Thus, the tool used to support the new methodology proposed is IDK as it is independent from the development process and it may be personalized for the application under development.

Table 1. Comparing Prometheus and INGENIAS

	Prometheus	**INGENIAS**
Proper development process	YES	NO: Based in the USDP (analysis and design phases)
General process to generate code from the models	NO: Only obtains code for JACK language	YES: Based in template definitions
Iterative development	YES	YES
Model-driven development (MDD)	NO: Only proposes a correspondence between design models and JACK code	YES
Requirements capture	YES: A version of KAOS is used to describe the system's goals [30] complemented with the description of scenarios that illustrate the operation of the system. In addition, in [5] guidelines appear to generate the artifacts of the Prometheus system specification from organizational models expressed in i*	YES: Performed by means of use case diagrams. Then, use cases are associated to system goals, and a goals analysis is performed to decompose them into easier ones; and finally tasks are associated to get the easiest goals
Meta-model	YES [6]	YES
Mechanisms to discover agents and interactions	YES: Groups functionalities through cohesion and coupling criteria	NO
Agent model	BDI-like agents	Agents with mental states

Table 2. Comparing PDT and IDK

	PDT	**IDK**
Supported methodology	Prometheus	INGENIAS
Interface references the development process	YES: Diagrams are grouped in three levels according to the three Prometheus phases	NO: Possibility to create packets that correspond to the diverse phases of the process. Models of each phase are added to the corresponding packet
Mechanisms to prioritize parts of the project	YES: Three scope levels (essential, conditional and optional) [26]	NO
Code generation	YES: JACK http://www.agent-software.com/	YES: JADE http://jade.tilab.com/
Report generation of the MAS specification in HTML	YES	YES
Model fragmenting in various pieces	NO: For instance, only one diagram may be created to in order to gather all the objectives of the system	YES
Save a diagram as an image	YES	YES
Deployment diagrams	NO	YES
Agent communication	Defined in basis of messages and interaction protocols. Does not use a specific communication language. For JACK, there is a module compliant with FIPA [32]	Defined in accordance with communication acts of the agent communication language (ACL) proposed by FIPA http://www.fipa.org/specs/fipa00061/
Utility to simulate MAS specifications before generating the final code	NO	YES: Realized on the JADE platform. It is possible to manage interaction and tasks, and to inspect and modify the agents' mental states

4 Phases of the New Methodology

First Phase. In the first stage of the methodology proposed an *analysis overview diagram* is created. This diagram is used to develop a high level view of the system requirements [29]. This diagram will specify, in main lines, which are the actors - entities (human or software/hardware) external to the system – that interact with our system, where the perceptions that enter the system come from, which are the responses of the system (actions), an initial proposal of which might be the system roles, what messages are sent, and some used data. This kind of diagram appeared for the first time in PDT version 2.5.

Second Phase. Prometheus defines a proper detailed process to specify, implement and test/debug agent-oriented software systems. This process incorporates three phases: (1) *system specification* identifies the basic goals and functionalities of the system, develops the use case scenarios that illustrate the functioning of the system, and specifies which are the inputs (percepts) and outputs (actions) – it obtains the scenarios diagram, goal overview diagram, and system roles diagram; (2) *architectural design* 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; and, (3) *detailed design* centers 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 to obtain the implementation in the agent-oriented programming language JACK.

The two first phases proposed in Prometheus (*system specification* and *architectural design*) are used to be the next phase of the new integrative methodology. The user identifies the agents and their interactions following the guidelines offered by Prometheus in these phases. In general terms, the mechanism provided by Prometheus to identify agents consists in identifying the goals during the system specification phase, and then in grouping the goals to obtain functionalities. Next, in the architectural design phase, functionalities are grouped to obtain the system agents, using cohesion and coupling criteria to decide which the best groupings are. These two concepts are essential in Software Engineering to obtain a good software development (the one that has maximum cohesion and minimum coupling) and to ease its further maintenance. On the other hand, the process for obtaining interactions between agents consists in changing the column role in the scenarios by the agent associated with each role. After that, if there are scenario steps with different agents then it means that there should be an interaction between the agents. Thus, the MAS developer gets an initial model according to Prometheus, following its two first stages (*system specification* and *architectural design*).

Next, an example of a moving robot application for the detection and following of humans is introduced in order to depict some diagrams obtained in the two Prometheus phases mentioned previously. In this application [14], a 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.

In *system specification* phase, after obtaining the goals and scenarios diagrams, the roles are identified by clustering goals and linking perceptions and actions (see Fig. 1). *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 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 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*).

One task carried out in *architectural design* 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 *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 2). After that agent conversations (interaction protocols - IP) are defined in order to describe what should happen to realize the specified goals and scenarios. For instance, *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). *Central_IP* protocol 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).

It has been shown in previous figures that there are entities, such as percepts and actions, 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.

Mapping Prometheus into INGENIAS. Afterwards, mappings are used to obtain an equivalent model in INGENIAS. From this point on the advantages offered by

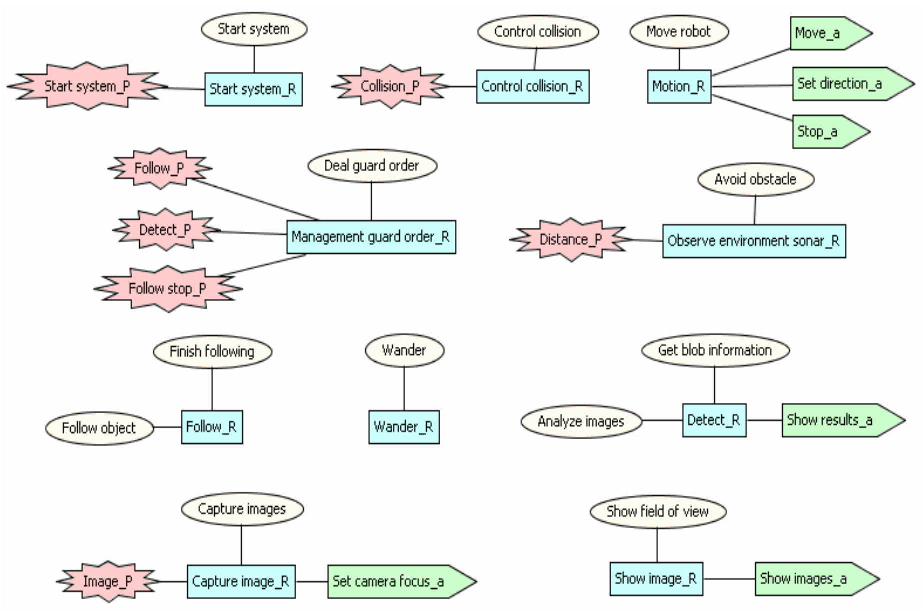


Fig. 1. System Roles Diagram

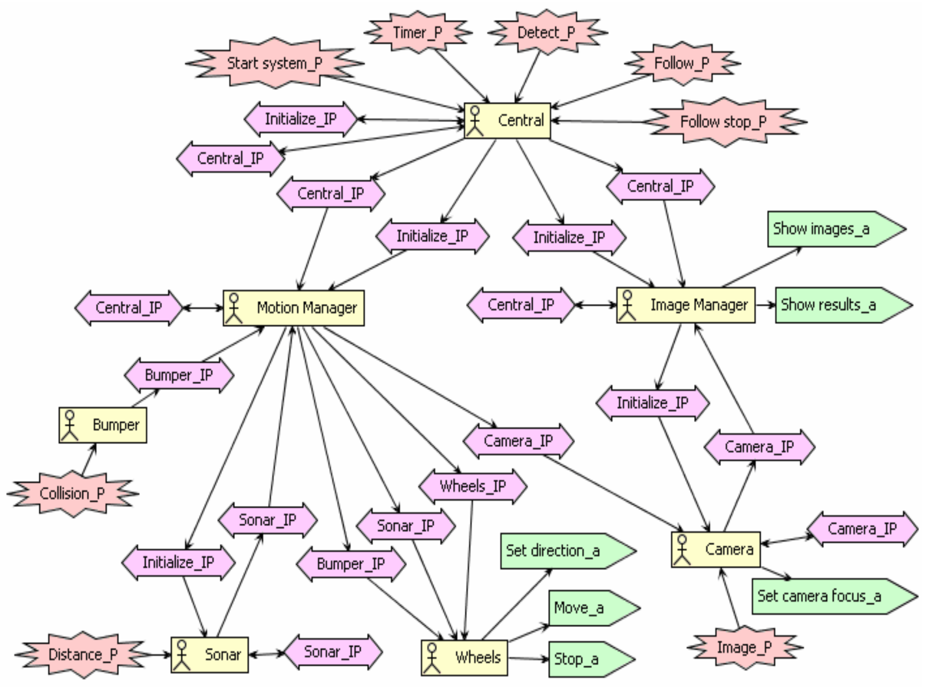


Fig. 2. System Overview Diagram

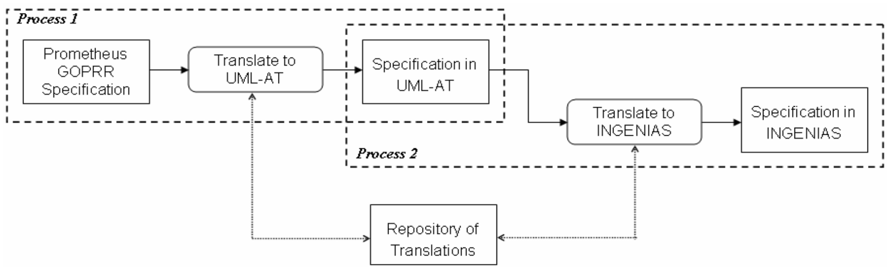

Process 1
Prometheus GOPRR Specification
Translate to UML-AT
Specification in UML-AT
Translate to INGENIAS
Specification in INGENIAS
Process 2
Repository of Translations

Fig. 3. Mapping Prometheus into INGENIAS

model-driven software development are used. The mappings are defined leaning in an intermediate language denominated Unified Modeling Language for Activity Theory (UML-AT) [8]. UML-AT allows establishing bidirectional transformations between models of different languages. There exists a previous experience in integrating two methodologies, Tropos and INGENIAS, using UML-AT [7].

The process of transforming Prometheus into INGENIAS methodology elements, with the help of intermediate language UML-AT, is shown in Fig. 1. In first place (process 1), we start from the Prometheus meta-model specified with language GOPRR (Graph Object Property Relationship Role) [20]. Translation rules to obtain elements, expressed in UML-AT language, equivalent to the ones selected in Prometheus, are created and used. Next (process 2), translation rules are used to obtain the specification in INGENIAS equivalent to the one obtained in UML-AT language. The Repository of Translations contains tuples indicating the matches and instantiation functions used in the translation, as well as the elements participating in it (either used in the process or created as a result of it) and an identifier of the specification to which each one belongs to. A match represents the translation between two sets of structures, the source pattern (it is described in the source language) and the target pattern (it is described in the target language). An instantiation function describes the correspondence of the variables in the source patterns with the elements in the current specifications according to the matching that is presented.

ATLAS Transformation Language (ATL) [2], a model transformation language compliant with the OMG MOF/QVT (Queries / Views /Transformation), can be used as an alternative to approach the problem of transforming Prometheus to INGENIAS. A meta-model and a model expressed in original language (in this case, Prometheus), a destination meta-model (in this case, INGENIAS) and rules defined with ATL to perform the transformation are needed in this case. The result is a destination model (INGENIAS, in this case) equivalent to the original model. The meta-models and models can be defined in Ecore, the language used by Eclipse Modeling Framework (EMF). The INGENIAS meta-model, defined initially in GOPRR, has been migrated to Ecore [9]. Thus, it is only necessary to define: (1) the Prometheus meta-model with Ecore, and, (2) the transformation rules in ATL. We have decided to use UML-AT because it is a technology related to the research group that has developed INGENIAS. In addition, it supposes the same service load as using ATL: to define a meta-model (for Prometheus in GOPRR) and translation rules (to transform a Prometheus specification into a UML-AT specification). The corresponding part to transform UML-AT into INGENIAS is solved in [8]. A tool called Activity Theory Assistant

(ATA) has been developed to help using the techniques based on the theory of the activity and to support the translation process. ATA is embedded in a plug-in of the IDK. Notice, however, that the current IDK version available in SourceForge, http://sourceforge.net/projects/ingenias, does not include it.

In Prometheus, in order to describe the interactions among agents, interaction protocols using a reviewed version of Agent UML (AUML) denominated AUML-2 are developed. UML-AT has already been applied to establish correspondences between FIPA protocols designed with AUML models [1] and INGENIAS models [8]. This work could be taken as departure point to transform interaction protocols obtained with Prometheus into the equivalent notation used in INGENIAS. The IDK tool, which provides support to INGENIAS, allows representing protocols according to the AUML annotation [15]. This means that the interaction protocols created with Prometheus could be used directly in INGENIAS, with no need to use any transformation. Nevertheless, its development has not evolved enough. In fact, in version 2.6 of the IDK this utility no longer appears.

At present, an informal approach is followed to transform Prometheus models into the equivalent INGENIAS models [12], [13]. The transformations are carried out manually. The next paragraphs explain how in the robot application some information obtained in Prometheus models are transformed into INGENIAS models.

A percept is a piece of information from the environment received by means of a sensor. In Prometheus, percepts must at least belong to one functionality, and, thus, to the agent associated to that functionality, too. The relations among percepts and roles (Percept → Role) and the relations among percepts and agents (Percept λ → Agent) appear in the system roles diagram and the system overview diagram, respectively. The percepts of a Prometheus agent can be modeled in INGENIAS by specifying a collection of operations in an application. In the INGENIAS environment model, an *EPerceivesNotification* relation between the agent and the corresponding application will be established. In a Prometheus percept descriptor, there is a field, Information carried, where it is specified the information transported as part of the percept. In INGENIAS, this information is included with an *ApplicationEventSlots* type of event associated to *EPerceivesNotification* relation. In the same way, in Prometheus, also every action must at least belong to one functionality and the agent associated to the functionality must execute it. The relations among actions and roles (Role → Action) and the relations among actions and agents (Agent → Action) appear in the system roles diagram and the system overview diagram, respectively. An action represents something that the agent does to interact with the environment. In INGENIAS, actions on the environment are assumed to be calls to operations defined in the applications. Therefore, an action present in Prometheus will be transformed into an application operation present in the environment model in INGENIAS. *EPerceives* will be used to establish the relation between the agent and the application.

In the environment model, an Agent - ApplicationBelongs To → Application relation will be also established to express that an agent uses an application. Fig. 4 shows the environment model obtained in INGENIAS to the *Camera* agent after applying the described equivalences.

A goal that appears in the goal overview diagram used in Prometheus will correspond to a goal in the goals and tasks model in INGENIAS. *AND* and *OR* dependencies between goals can be established in both models; therefore, it is possible

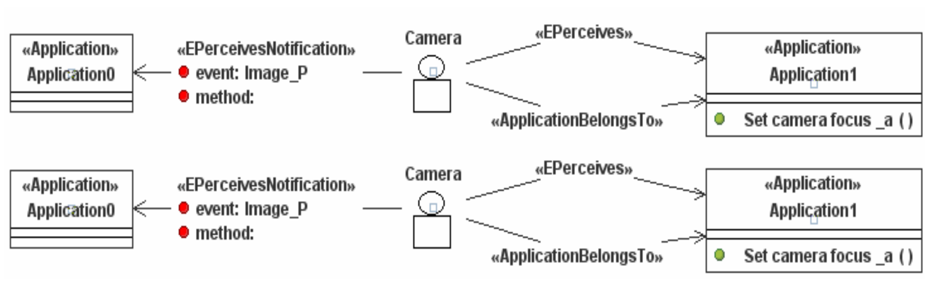


Fig. 4. Environment model for the *Camera* agent

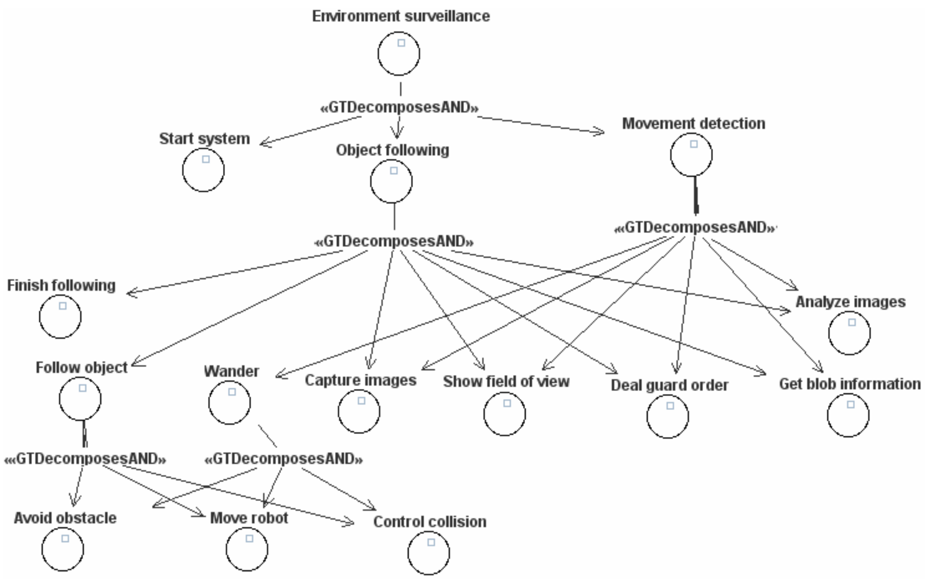


Fig. 5. Goals in the goals and tasks model

to directly transfer these relations from one model to another. In the goals and tasks model, a *GTDecomposeAND* relation and a *GTDecomposeOR* relation will be established to reflect an *AND* and *OR* relation between goals, respectively. When a goal has only one sub-goal, *GTDecomposes* is used. Fig. 5 summarizes the goals that appear in INGENIAS goals and tasks model and that have been obtained from the Prometheus goal overview diagram.

In the system roles diagram of Prometheus methodology, relations between goals and functionalities are established. The latter will be grouped to determine the types of agents in the system - the relation among agents and roles, Agent → Role, appear in the agent-role grouping diagram, whilst the relation among roles and goals, Role → Goal, appear in the system roles diagram. Therefore, implicitly, there is a relation between goals and agents. So, there is a Agent - *GTPursues* → Goal relation in the agent model. IDK only supports roles to generate the code that corresponds to an interaction. Therefore, for every agent identified in Prometheus, an associated role is

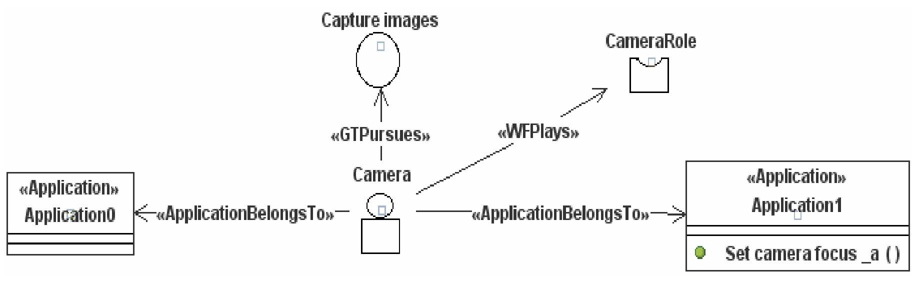


Fig. 6. Fragment of *Camera* agent model

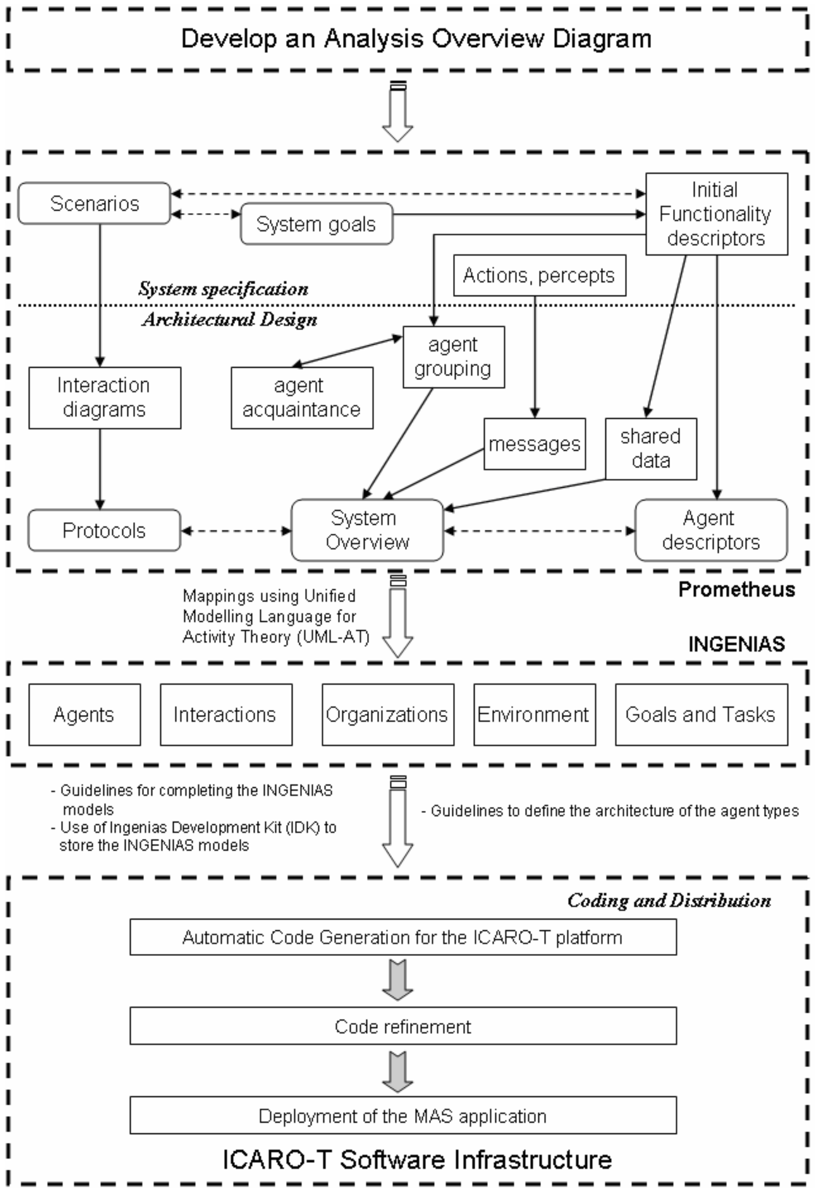
necessary to state its participation in an interaction. So, we will establish an Agent - *WFPlays* → Role relation. Fig. 6 depicts a fragment of the INGENIAS agent model corresponding to the *Camera* agent.

Third Phase. The new methodology does not reuse the last phase of Prometheus (*detailed design*) because it is too much centered in BDI-like agents. Moreover, Prometheus also describes how obtained entities are transformed in the design phase into the concepts used for a specific implementation language (JACK). These two aspects, centering in a single type of agent and defining a mechanism to generate code for a particular implementation language, suppose, in principle, a loss of generality. In the new methodology, once the equivalent model in INGENIAS has been obtained, the architecture of each type of identified agent is provided. The possible types of agents are the ones available in ICARO-T: cognitive agents and reactive agents. In this phase the necessary guidelines for completing all the INGENIAS models already exist.

Fourth Phase. With respect to code generation, the INGENIAS proposal is followed. INGENIAS generalizes a process to transform the models, generated in the phase of design, in running code for any destination platform [7]. It is based in the definition of templates for each destination platform and procedures for extracting information present in the models. Once the code has been obtained, the developer refines the resulting code completing any information that was not contained in the specifications (models) or in the templates. Finally, the application is deployed.

ICARO-T is the platform selected for running the agents [11], [25]. It offers four categories of reusable component models: *agent organization models* to describe the overall structure of the system, *agent models*, *resource models* to encapsulate computing entities providing services to agents, and *basic computing entities*. There are several reasons for selecting the multi-agent platform ICARO-T. The use of its components has allowed to significantly reducing time and effort in the design and implementation phases by an average of a 65 percent. In the phases of testing and correction cycles the errors are also reduced. Consequently, the applications require less resources and lower implementation time [25].

The ICARO-T components have been used in Spanish telecommunications company Telefonica for developing several voice recognition services. At the moment, it is also being used by other research teams. This is the case, for example, in an e-learning project denominated ENLACE [31]. Fig. 7 shows the technology and tools used in the integrative methodology proposed.



Develop an Analysis Overview Diagram
Scenarios
System goals
Initial Functionality descriptors
Actions, percepts
System specification
Architectural Design
agent grouping
agent acquaintance
Interaction diagrams
messages
shared data
Protocols
System Overview
Agent descriptors
Prometheus
Mappings using Unified Modelling Language for Activity Theory (UML-AT)
INGENIAS
Agents
Interactions
Organizations
Environment
Goals and Tasks
- Guidelines for completing the INGENIAS models
- Use of Ingenias Development Kit (IDK) to store the INGENIAS models
- Guidelines to define the architecture of the agent types
Coding and Distribution
Automatic Code Generation for the ICARO-T platform
Code refinement
Deployment of the MAS application
ICARO-T Software Infrastructure

Fig. 7. Multi-agent system development methodology

5 Conclusions and Future Work

The combination of current technologies (Prometheus, INGENIAS, UML-AT and ICARO-T) has given rise to a new integrative methodology for the development of agent-based systems. It uses the guidelines offered by Prometheus to identify agents and their interactions. Later, the obtained model, following Prometheus methodology, is transformed into INGENIAS to continue the development. This transformation is performed under UML-AT. Once modeling has ended up, code is generated for the ICARO-T platform.

In order to use this methodology definitively it is necessary: (1) to specify the Prometheus meta-model in GOPRR, (2) to create the rules to translate Prometheus concepts in UML-AT language, and, (3) to define templates for the IDK to generate code for the ICARO-T platform. A first proposal in order to transform Prometheus models into equivalent INGENIAS models using an informal language has recently been considered [12].

Acknowledgements. 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. Bauer, B., Odell, J.: UML 2.0 and agents: How to build with the new UML standard. Journal of Engineering Applications in Artificial Intelligence 18(2), 141–157 (2005)
2. Bézivin, J., Jouault, F., Touzet, D.: An introduction to the ATLAS model management architecture. Research Report LINA (05-01) (2005)
3. Cabri, G., Puviani, M., Leonardi, L.: The MAR&A methodology to develop agent system. In: First International Conference on Agents and Artificial Intelligence, pp. 501–506 (2009)
4. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: From standardisation to research. International Journal of Agent-Oriented Software Engineering 1(1), 91–121 (2007)
5. Cysneiros, G., Zisman, A.: Refining Prometheus methodology with i*. In: Third International Workshop on Agent-Oriented Methodologies (2004)
6. Dam, K.H., Winikoff, M., Padgham, L.: An agent-oriented approach to change propagation in software evolution. In: Australian Software Engineering Conference, pp. 309–318 (2006)
7. Fuentes, R., Gómez-Sanz, J.J., Pavón, J.: Integrating agent-oriented methodologies with UML-AT. In: Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1303–1310 (2006)
8. Fuentes, R., Gómez-Sanz, J.J., Pavón, J.: Model integration in agent-oriented development. International Journal of Agent-Oriented Software Engineering 1(1), 2–27 (2007)
9. García-Magariño, I., Gómez-Sanz, J.J., Pavón, J.: Representación de las relaciones en los metamodelos con el lenguaje Ecore. In: Desarrollo de Software Dirigido por Modelos, DSDM 2007 (2007)
10. Garcia-Magariño, I., Gómez-Sanz, J., Fuentes, R.: INGENIAS development assisted with model transformation By-Example: A practical case. In: Seventh International Conference on Practical Applications of Agents and Multi-Agent Systems (2009)
11. Garijo, F., Polo, F., Spina, D., Rodríguez, C.: ICARO-T User Manual. Internal Report, Telefonica I+D (May 2008)
12. Gascueña, J.M., Fernández-Caballero, A.: Prometheus and INGENIAS agent methodologies: A complementary approach. In: Luck, M., Gomez-Sanz, J.J. (eds.) Agent-Oriented Software Engineering IX. LNCS, vol. 5386, pp. 131–144. Springer, Heidelberg (2009)
13. Gascueña, J.M., Fernández-Caballero, A.: Towards an integrative methodology for developing multi-agent systems. In: International Conference on Agents and Artificial Intelligence (2009)

14. Gascueña, J.M., Fernández-Caballero, A.: Agent-based modeling of a mobile robot to detect and follow humans. In: Håkansson, A., Nguyen, N.T., Hartung, R.L., Howlett, R.J., Jain, L.C. (eds.) Agent and Multi-Agent Systems: Technologies and Applications. LNCS, vol. 5559, pp. 80–89. Springer, Heidelberg (2009)
15. Gómez Sanz, J.J., Pavón, J.: INGENIAS Development Kit (IDK) Manual. Version 2.5.2 (2008), http://heanet.dl.sourceforge.net/sourceforge/ingenias/ingeniasmanual.v2.5.pdf
16. Hahn, C., Madrigal-Mora, C., Fischer, K.: A platform-independent metamodel for multi-agent systems. Autonomous Agents and Multi-Agent Systems 18(2), 239–266 (2009)
17. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading (1999)
18. Jarraya, T., Guessoum, Z.: Towards a model driven process for multi-agent system. In: Burkhard, H.-D., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS 2007. LNCS (LNAI), vol. 4696, pp. 256–265. Springer, Heidelberg (2007)
19. Jennings, N.R., Wooldridge, M.: Applying agent technology. International Journal of Applied Artificial Intelligence 9(4), 351–359 (1995)
20. Kelly, S., Lyytinen, K.S., Rossi, M.: METAEDIT+ - A fully configurable multi-user and multi-tool CASE and CAME environment. In: Constantopoulos, P., Vassiliou, Y., Mylopoulos, J. (eds.) CAiSE 1996. LNCS, vol. 1080, pp. 1–21. Springer, Heidelberg (1996)
21. Padgham, L., Winikoff, M.: Developing Intelligent Agents Systems: A Practical Guide. John Wiley and Sons, Chichester (2004)
22. Padgham, L., Thangarajah, J., Paul, P.: Prometheus Design Tool. Version 2.5. User Manual (2008), http://www.cs.rmit.edu.au/agents/pdt/docs/PDT-Manual.pdf
23. Pavón, J., Gómez-Sanz, J.J., Fuentes, R.: The INGENIAS methodology and tools. Agent-Oriented Methodologies. Idea Group Publishing, USA (2005)
24. Pavón, J., Gómez-Sanz, J.J., Fuentes, R.: Model driven development of multi-agent systems. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 284–298. Springer, Heidelberg (2006)
25. Pavón, J., Garijo, F., Gómez-Sanz, J.: Complex systems and agent-oriented software engineering. In: Weyns, D., Brueckner, S.A., Demazeau, Y. (eds.) EEMMAS 2007. LNCS (LNAI), vol. 5049, pp. 3–16. Springer, Heidelberg (2008)
26. Perepletchikov, M., Padgham, L.: Systematic incremental development of agent systems, using Prometheus. In: Fifth International Conference on Quality Software, pp. 413–418 (2005)
27. Perini, A., Susi, A.: Automating model transformations in agent-oriented modelling. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 167–178. Springer, Heidelberg (2006)
28. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer 39(2), 25–31 (2006)
29. 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 (LNAI), vol. 4953, pp. 63–72. Springer, Heidelberg (2008)
30. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Fifth IEEE International Symposium on Requirements Engineering, pp. 249–263 (2001)
31. Verdejo, M.F., Celorrio, C.: A multi-agent based system for activity configuration and personalization in a pervasive learning framework. In: Third IEEE International Workshop on Pervasive Learning, pp. 177–181 (2007)
32. Yoshimura, K.: FIPA JACK: A plugin for JACK Intelligent AgentsTM. Technical Report, RMIT University (2003)