Prometheus and INGENIAS Agent Methodologies: A Complementary Approach

José M. Gascueña and Antonio Fernández-Caballero

Departamento de Sistemas Informáticos
Instituto de Investigación en Informática de Albacete (I3A)
Universidad de Castilla-La Mancha, 02071-Albacete, Spain
{jmanuel,caballer}@dsi.uclm.es

Abstract. A great number of methodologies to develop multi-agent systems (MAS) have been proposed in the last few years. But a unique methodology cannot be general enough to be useful for everyone without some level of customization. According to our knowledge, existent agent-based surveillance systems have been developed ad-hoc and no methodology has been followed. We are interested in creating tools that allow to model and to generate monitoring environments. This has motivated the selection of Prometheus and INGENIAS methodologies, to take advantage of both approaches in developing agent-based applications. In this paper a collection of equivalences between the concepts used in both methodologies is described extensively.

1 Introduction

The use of surveillance systems has grown exponentially during the last decade, and has been applied in many different environments [29]. A distributed configuration is mandatory to get scalable and robust surveillance applications ([28], [27], [18], [22]). These systems are complex and work in highly dynamic environments, where scattered sensors (e.g., camera, temperature, presence detection, and microphone) can decide and act with some degree of autonomy, and cooperate and coordinate for complete tracking of special situations. These characteristics are often cited as a rationale for adopting agent technology [31]. In fact, this technology has already been used in several surveillance systems (e.g. [26], [16], or [1]). According to our knowledge, existent agent-based surveillance systems have been developed ad-hoc and no methodology has been followed. Nevertheless, using a methodology allows to share the same terminology, annotation, models, and development process [2].

A great number of methodologies to develop multi-agent systems (MAS) have been proposed in the last few years (e.g. Gaia, Tropos, Message, MaSE). But a unique methodology cannot be general enough to be useful for everyone without some level of customization [4]. Habitually, techniques and tools proposed in different methodologies are combined.

M. Luck and J.J. Gomez-Sanz (Eds.): AOSE 2008, LNCS 5386, pp. 131–144, 2009.

2 Combining Prometheus and INGENIAS

In [10] several agent-oriented software development methodologies are presented, an evaluation and comparison of this methodologies is carried out by Tran and Low (chapter 12), and Henderson-Sellers introduce in chapter 13 a conceptual framework that enables reusing methodology fragments to create a specific methodology for each project faced. Some works on agent methodology integration are related to Tropos to INGENIAS mapping [7], or FIPA protocols specified in AUML transformed in equivalent INGENIAS models [8]. Also, the CAFnE toolkit [3] provides a framework that facilitates domain experts in making modifications to a deployed agent-based system without the assistance of agent programmers. Recently, a proposal for facilitating MAS complete life cycle through the Protégé-Prometheus approach has been presented [24].

In our case, we shall reuse parts of Prometheus and INGENIAS methodologies due to the following reasons. Once both methodologies have been studied, we have observed that the process followed in INGENIAS [17] during the analysis and design phases of MAS is very complex and difficult to follow, because is it not clear how the different models are being constructed along the phases, despite the documented general guidelines. In addition, it does not offer guidelines helping to determine which the elements that form the MAS are. It is solely the experience of the developer that determines its identification. On the contrary, Prometheus does offer these guidelines. These guidelines are also able to 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 of the agent-based application. In addition, Prometheus is also useful because it explicitly considers agent perceptions and actions as modeling elements. INGENIAS (1) is currently focused to model-driven development (MDD) [18], (2) offers a process to generate code for any agent platform [8], and, (3) is supported by INGENIAS Development Kit (IDK) [17] that can be personalized for any application domain. These three characteristics are not offered by Prometheus. Nowadays the use of model-driven engineering (MDE) techniques along the software development life cycle is gaining more and more interest [23]. An MDD has 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.

3 In-depth Comparison of Prometheus and INGENIAS

Prometheus [13] defines a 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 help better understanding 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 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. 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 centers on developing the internal structure of each agent and how each agent will perform his 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); this supposes, in principle, a loss of generality. The debugging mechanisms used in Prometheus are described extensively in [15]. The Prometheus methodology is supported by Prometheus Design Tool (PDT) [25].

On the other hand, the foundation of INGENIAS is the definition of MAS meta-model and a set of MDD tools oriented towards agents (model edition, verification, validation and transformation) integrated in INGENIAS Development Kit (IDK). The meta-model and IDK can be customized for a specific application domains. The meta-model describes the elements that enable modeling MAS from different points of view - agent, organization, environment, goals and tasks, and interaction [17]. The agent perspective considers the elements to specify the behavior of each agent. The organization perspective shows the system architecture. From a structural point of view, the organization is a set of entities with relationship of aggregation and inheritance. It defines a schema where agents, resources, tasks and goals may exist. Under this perspective, groups may be used to decompose the organization, plans, and workflows to establish the way the resources are assigned, which tasks are necessary to achieve a goal, and who has the responsibility of carrying them out. The environment perspective defines the agents' sensors and actuators, and identifies the system resources and

Table 1. Comparing Prometheus and INGENIAS

	Prometheus	**INGENIAS**
Proper development process	YES	NO: Based in the USDP (analysis and design phases). An agile process is suggested in [19]
General process to generate code from the models	NO: Only obtains code for JACK language	YES: Based in template definitions
Iterative development process	YES	YES
Model-driven development (MDD)	NO: Only proposes a correspondence between design models and JACK code	YES
Requirements capture	YES	YES
Meta-model	YES [6]	YES
Mechanisms to discover agents and interactions among agents	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) [21]	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: Format .png. The image resolution can be configured	YES: Formats .jpg, .svg, .png, wbmp, .bmp and .jpeg
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/fipa*00061/
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
Plans Executable as stand-alone	Textual description YES	Graphical description YES
Integration in Eclipse	YES: See [14]	YES: in IDK version 2.7

applications. The goals and tasks perspective describes the relations between tasks and goals. The interaction perspective describes how the coordination among the agents is produced. INGENIAS, at difference with Prometheus, does not define its own development process; rather it adopts the unified software development process (USDP) [11] as a guideline to define the steps necessary to develop the elements and diagrams of MAS during the analysis and design phases of the USDP. Moreover, INGENIAS facilitates a general process to transform the models generated during the design phase into executable code for all destination platforms. This general process is based in the definition of templates for each destination platform and procedures for extraction of information present in the models.

Both Prometheus and INGENIAS methodologies support the facility to capture requirements. In the Prometheus system specification phase, 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*. In INGENIAS, requirements capture is 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.

In summary, 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) and INGENIAS Development Kit (IDK) 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. The tool that it will use to support the new methodology is IDK as it is independent from the development process and it may be personalized for the application under development.

4 Mapping Prometheus into INGENIAS

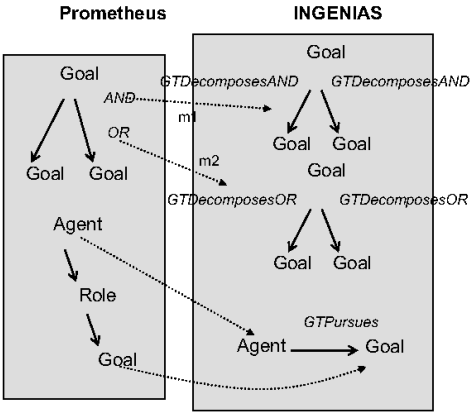
In the new methodology, we propose that the MAS developer firstly follows the system specification and architectural design phases proposed in Prometheus. Thus, an initial model in accordance with Prometheus is obtained using the guidelines that enable to identify the agents and their interactions. Afterwards, an equivalent model in INGENIAS is obtained using the collection of equivalences (mappings) between the concepts used in both methodologies. Next, modeling goes on with INGENIAS; thus, benefiting from the advantages of model-driven software development.

The detailed design phase of Prometheus is not followed because it mentions a specific agent-based model, namely the BDI model, which is different from the mental state agents used in INGENIAS. In this section, the collection of equivalences is described. We would like to point out that we are only describing the mapping from Prometheus into INGENIAS of the concepts used during the system specification and architectural design phases defined in Prometheus.

The proposed mappings have mainly been deduced on the basis of the organizations and relations between organizations that are possible to create with the supported tools PDT and IDK, respectively. In our next exposition, we use the Entity 1 - *RelationX* $\rightarrow$ Entity 2 notation to express that Entity 1 and Entity 2 are related through relation *RelationX*. The direction in which the arrow is pointing accurately reflects what the graphic representation of the relation is like. In the figures, dotted arrows are used to stand out how the organizations of Prometheus are transformed into the equivalent organizations of INGENIAS. In the tables information is offered on the models in which the structures represented graphically appear; and, textually, the transformations carried out are described.

4.1 Mapping Prometheus Goals

There are different approaches for the use of the term "goal" [9]: (1) in classical planning, it is seen as a description of the state of the world to be



Prometheus	INGENIAS
Goal overview diagram Goal AND dependences between goals OR dependencies	Goals and tasks model Goal *GTDecomposesAND* relation *GTDecomposesOR* relation *GTDecomposes* if a goal has only one subgoal
Agent → Role → Goal	Organization model Agent model Goal and task model Agent → *GTPursues* → Goal

Fig. 1. Mapping information related with Prometheus goals into INGENIAS

reached - goals as aggregation; (2) in the BDI model, it is a wish to be satisfied - goals as entities; and, (3) to reflect the requirements that the system must fulfill in the design - goals as requirements. In INGENIAS, the goals are initially taken as self-representing entities (goals-as-entities approach) which guide the behavior of the agent. To take into account the planning approach (goals-as-aggregation approach), the goals must be allowed to connect with the set of elements (e.g. predicates) that they represent. As for the last approach (goals-as-requirements approach), the assimilation of goals with requirements is purely interpretative. It is up to the engineer to consider a goal as a requirement or not.

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 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. The arrows m1 and m2 of Fig. 1 highlight the transformation of the structures between goals in Prometheus into the equivalent structures in INGENIAS - m1 and m2 show the transformation of AND and OR structures, respectively. When a goal has only one sub-goal, *GTDecomposes* is used.

In the system roles[1] 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. In INGENIAS, one of the consistency criteria of the goals and tasks meta-model expresses that *"the goal that appears in a goals and tasks model must appear in an agent model or in an organization model"* (criterion 2). Basing ourselves on the information from the system roles diagram and taking into account the previous comments, for each goal, the following relations[2] will be established in INGENIAS: (a) Agent - *GTPursues* → Goal in the organization model, (b) Agent - *GTPursues* → Goal in the agent model and (c) Agent - *GTPursues* → Goal in the tasks and goals model. All these equivalencies are summarized graphically and textually in Fig. 1.

4.2 Mapping Prometheus Agents

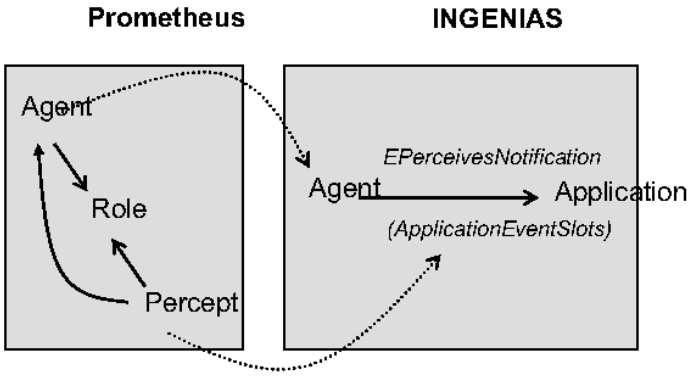
Every agent identified in the Prometheus methodology is reflected in INGENIAS in the agent model and in the organization model, based on the agent model consistency criterion *"for every agent in the organization model, there must be an instance for the agent model and vice-versa"* (criterion 4). If the agent must perceive changes in the environment, it will be also shown in the environment model. The agents that interact with other agents should also be represented in the interaction model. However, IDK only supports roles to generate the code that corresponds to an interaction. Therefore, for every agent identified in Prometheus, an associated role in the corresponding interaction model in INGENIAS to state its participation in the interaction has to be created. Likewise, we will establish an Agent - *WFPlays* → Role relation in the organization or agent model. Finally, we should remember that in the goals and tasks model, agents will be also obtained, according to what was specified in section 4.1.

4.3 Mapping Prometheus Percepts and Actions

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 → Agent) appear in the system roles diagram and the system overview diagram, respectively. The percepts of a Prometheus agent can be modeled in

[1] Sometimes, in the description of the mappings, the term role is used instead of the term functionality. Role is the term used in PDT, whereas functionality is the term used in Prometheus.

[2] What we really mean is that instances of the corresponding entities will be created and will be related through the pertinent relation.



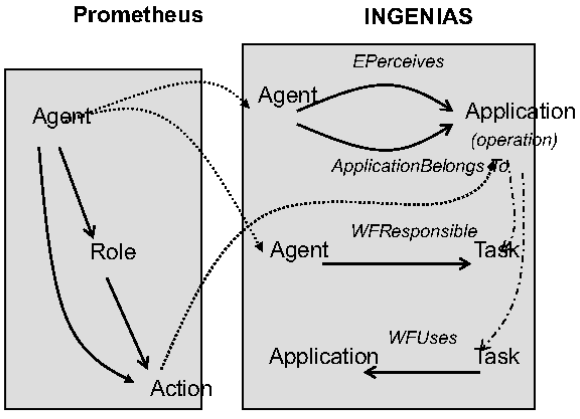
Prometheus	INGENIAS
Agent → Role ← Percept	Environment model Agent – *EPerceivesNotification* → Application *ApplicationEventSlots* type of event associated to *EPerceivesNotification* relation

Fig. 2. Mapping information related with Prometheus percepts into INGENIAS

INGENIAS by specifying a collection of operations in an application. Depending on whether the application existed prior to the MAS development or was developed ad-hoc, for this purpose, we can specify the application in an environment application or an internal application, represented by *EnvironmentApplication* and *InternalApplication*, respectively.

The consistency criterion 2 of the environment model states that *"every agent that perceives changes in the environment must appear in the environment model associated to an application"*. Therefore, in the 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. The basic ingredients of these equivalencies are graphically and textually summarized in Fig. 2.

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



Prometheus	INGENIAS
Agent → Role → Action	Environment model Agent – *EPerceives* → Application Agent – *ApplicationBelongs To* → *Application* Agent model Agent – *WFResponsible* → Task Task – *WFUses* → Application Organization model Agent – *WFResponsible* → Task

Application can be refined in EnvironmentApplication or InternalApplication

Fig. 3. Mapping information related with Prometheus actions into INGENIAS

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. In addition, the operations in INGENIAS will be executed by the corresponding agent, by means of the relevant task. Thus, in the INGENIAS agent model, it will be created (1) an Agent - *WFResponsible* → Task relation to specify the agent responsible for carrying out the task, which triggers the execution of an action on the environment; and (2) a Task - *WFUses* → Application relation to express that an application is used in the task. If it is decided to specify the application in an environment or internal application, Task - *WFUses* → Environment Application and Task - *WFUses* → Internal Application is choosed, respectively. On the other hand, according to the INGENIAS agent model consistency criterion 1, *"every task associated to an agent must appear in the organization model, indicating its role within the task global structure"*, in the organization model, Agent - *WFResponsible* Task will appear. This information is summarized in Fig. 3.

4.4 Mapping Prometheus Data

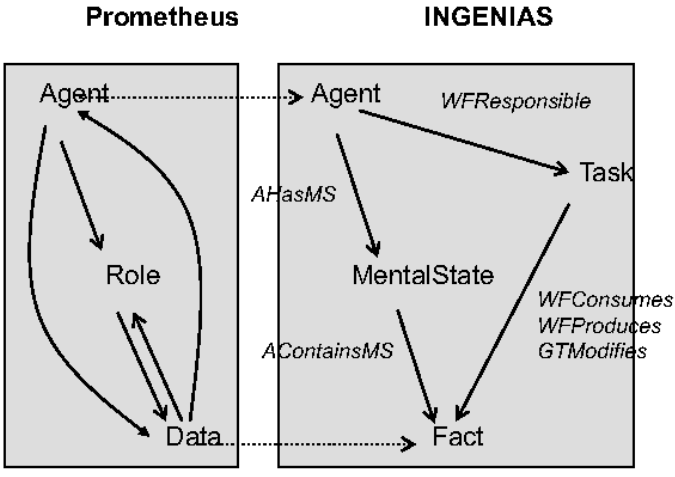
In INGENIAS, facts reflect information that is inherently true; for example, "water evaporates by applying heat", or any other information resulting from the execution of tasks. The data written and read by Prometheus agents will be made to correspond with facts or framefacts in INGENIAS. A framefact is a fact whose information is within its slots.

In the Prometheus system overview diagram, there are Agent → Data (expresses that the data is written by the agent) and Data → Agent (expresses that the data is read by the agent) relations. These structures would be translated to the INGENIAS agent model through the following procedure: a MentalState - *AContainsME* → Fact relation is created to specify a fact associated to a mental state, and an Agent - *AHasMS* → MentalState relation to specify that the mental state corresponds to the agent equivalent to the one we had in Prometheus. That is to say, an Agent - *AHasMS* → Mental State - *AContainsMS* → Fact structure will be get. The fact would become a Framefact instead of a Fact, if it includes more than one field of information. In INGENIAS, mental states are represented in terms of goals, tasks, facts, or any other entity that helps in state description. In INGENIAS goal and task model, Task - *WFConsumes* → Fact, Task - *WFProduces* → Fact, and Task - *GTModifies* → Fact relations are created to indicate that a fact is read, written and modified when executing a task, respectively. An agent will be responsible for executing that task - it is represented by an Agent - *WFResponsible* → Task relation. All these equivalencies are summarized graphically and textually in Fig. 4.

4.5 Mapping Prometheus Interaction Protocols

Prometheus offers a mechanism to derive interaction diagrams, and, as a result, interaction protocols from the scenarios developed. In one interaction protocol, there are agents that participate in the interaction and the messages that the agents send to one another.

In INGENIAS, the interaction model is used for describing how coordination between agents comes about. For each interaction protocol in Prometheus, we will develop an interaction model which will include: (1) an Interaction entity having the same name as the interaction protocol in Prometheus, (2) roles (from INGENIAS) associated to the agents that intervene in the interaction protocol, (3) an Interaction - *IInitiates* → Role relation, for the role associated to the agent that initiates the interaction protocol, (4) Interaction - *ICollaborates* → Role relations, for the roles associated to the agents (different to the initiating agent) that participate in the interaction protocol, and, (5) an Interaction - *IHasSpec* → GRASIASpecification relation. The GRASIASpecification entity will be associated to an interaction model where the messages sequence that intervenes in the interaction protocol, the agents that send them and their tasks involved will be described. In INGENIAS, the term Interaction Units, instead of messages, is used. In Prometheus, a protocol descriptor makes reference to the scenarios which include the protocol. When a scenario is created with PDT,



Prometheus	INGENIAS
	Agent model MentalState – *AContainsMS* → Fact Agent – *AHasMS* → MentalState
Agent → Role ← → Data Agent ← → Data	Goal and task model Agent – *WFResponsible* → Task Task – *WFConsumes* → Fact if Data is read Task – *WFProduces* → Fact if Data is written Task – *GTModifies* → Fact if Data is modified

Fig. 4. Mapping information related with Prometheus data into INGENIAS

a goal associated to the scenario is automatically generated. This is due to the contributions by Perepletchikov [21]. Therefore, the mentioned goal would be related to the interaction created in the INGENIAS interaction model.

5 Conclusions

After carrying out a comparative analysis of the Prometheus and INGENIAS methodologies, we realized that we can benefit from the simple guidelines offered by Prometheus in its development process to obtain an initial model of the MAS that we will be dealing with. Subsequently, we can move it into INGENIAS to proceed with the modeling, in order to benefit from the model-driven development. In order to make the transformations of concepts used in Prometheus into concepts used in INGENIAS the described mappings are applied.

At the moment these transformations are made manually. In order to implement our mapping proposal we plan to use some model transformation language. UML-AT [8] enables integrating N methodologies; but, as we are integrating only two, it is sufficient to use ATL (Atlas Transformation Language) [12]. In the foreseen implementation the INGENIAS Ecore meta-model will be used and the Prometheus Ecore meta-model will be created. Moreover, we are interested in going on adapting the IDK editor to model surveillance systems because it is independent from the development process and it may be personalized for the application under development.

Acknowledgements

This work is supported in part by the Junta de Comunidades de Castilla-La Mancha PBI06-0099 grant and the Spanish Ministerio de Educación y Ciencia TIN2007-67586-C02-02 grant.

References

1. Aguilar-Ponce, R., Kumar, A., Tecpanecatl-Xihuitl, J.L., Bayoumi, M.: A network of sensor-based framework for automated visual surveillance. Journal of Network and Computer Applications 30, 1244–1271 (2007)
2. 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)
3. Buddhinath Jayatilleke, G., Padgham, L., Winikoff, M.: A model driven development toolkit for domain experts to modify agent based systems. In: Padgham, L., Zambonelli, F. (eds.) AOSE VII / AOSE 2006. LNCS, vol. 4405, pp. 190–207. Springer, Heidelberg (2007)
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: 3rd 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: Proceedings of the Australian Software Engineering Conference, pp. 309–318 (2006)
7. Fuentes, R., Gomez-Sanz, J.J., Pavón, J.: Integrating agent-oriented methodologies with UML-AT. In: Proceedings of the Fifth international Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1303–1310 (2006)
8. Fuentes, R., Gomez-Sanz, J.J., Pavón, J.: Model integration in agent-oriented development. International Journal of Agent-Oriented Software Engineering 1(1), 2–27 (2007)
9. Gómez Sanz, J.J.: Modelado de sistemas multiagente. Ph.D thesis, Departamento de Sistemas Informáticos y Programación. Universidad Complutense de Madrid (2002)
10. Henderson-Sellers, B., Giorgini, P.: Agent-Oriented Methodologies. Idea Group Publishing, USA (2005)

11. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading (1999)
12. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
13. Padgham, L., Winikoff, M.: Developing intelligent agents systems: A practical guide. John Wiley and Sons, Chichester (2004)
14. Padgham, L., Thangarajah, J., Winikoff, M.: Tool support for agent development using the Prometheus methodology. In: First International Workshop on Integration of Software Engineering and Agent Technology, pp. 383–388 (2005)
15. Padgham, L., Winikoff, M., Poutakidis, D.: Adding debugging support to the Prometheus methodology. Engineering Applications of Artificial Intelligence 18(2), 173–190 (2005)
16. Patricio, M.A., Carbó, J., Pérez, O., García, J., Molina, J.M.: Multi-agent framework in visual sensor networks. EURASIP Journal on Advances in Signal Processing, Article ID 98639 (2007)
17. Pavón, J., Gomez-Sanz, J.J., Fuentes, R.: The INGENIAS methodology and tools. In: Agent-Oriented Methodologies. Idea Group Publishing, USA (2005)
18. 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)
19. Pavón, J.: INGENIAS: Développement Dirigé par Modèles des Systémes Multi-Agents. Habilitation à diriger des recherches de l'Université Pierre et Marie Curie (2006)
20. Pavón, J., Gomez-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)
21. Perepletchikov, M., Padgham, L.: Systematic incremental development of agent systems, using Prometheus. In: Fifth International Conference on Quality Software, pp. 413–418 (2005)
22. Remagnino, P., Shihab, A.I., Jones, G.A.: Distributed intelligence for multi-camera visual surveillance. Pattern Recognition 37(4), 675–689 (2004)
23. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer 39(2), 25–31 (2006)
24. 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)
25. Thangarajah, J., Padgham, L., Winikoff, M.: Prometheus Design Tool. In: Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems, pp. 127–128 (2005)
26. Ukita, N., Matsuyama, T.: Real-time cooperative multi-target tracking by communicating active vision agents. Computer Vision and Image Understanding 97(2), 137–179 (2005)
27. Valencia-Jiménez, J.J., Fernández-Caballero, A.: Holonic multi-agent systems to integrate independent multi-sensor platforms in complex surveillance. In: IEEE International Conference on Advanced Video and Signal based Surveillance, vol. 49 (2006)
28. Valencia-Jiménez, J.J., Fernández-Caballero, A.: Holonic multi-agent system model for fuzzy automatic speech / speaker recognition. In: Nguyen, N.T., Jo, G.S., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2008. LNCS, vol. 4953, pp. 73–82. Springer, Heidelberg (2008)

29. Valera, M., Velastin, S.A.: A review of the state-of-the-art in distributed surveillance systems. In: Intelligent Distributed Video Surveillance Systems. IEE Professional Applications of Computing Series, vol. 5, pp. 1–30 (2006)
30. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering, pp. 249–263 (2001)
31. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review 10(2), 115–152 (1995)
32. Yoshimura, K.: FIPA JACK: A plugin for JACK Intelligent Agents^{TM}. Technical Report, RMIT University (2003)