2

# A Multi-Agent System Architecture for Sensor Networks

María Guijarro, Rubén Fuentes-Fernández and Gonzalo Pajares
*University Complutense Of Madrid, Madrid, Spain*

## 1. Introduction

Today it is increasingly important a good design architectures that support sensors. This chapter shows how the design of the control systems for sensor networks presents important challenges because of using sensor networks has design problems. Besides the traditional aspects about how to process the data to get the target information, engineers need to consider additional aspects such as the heterogeneity and high number of sensors, and the flexibility of these networks regarding topologies and sensors in them.
The increasing availability of sensors plugable in networks at low costs is rapidly increasing their use for different applications like smart spaces or surveillance systems (Tubaishat, M. & Madria, S. 2003).These networks pose important challenges for engineers working in the development of the related control systems. Some of the most relevant are:

- *Potential high number of nodes*. The current trend is to set up networks densely populated with sensors and a minor number of controllers (Yick et al., 2008). These magnitudes imply that engineers must consider issues such as the organization of the communications and local pre-processing of data to save bandwidth and get suitable response times.
- *Sensor heterogeneity*. These networks include a wide variety of types of devices (e.g. cameras, motion sensors or microphones) whose management and usage differs (Hill et al., 2000). These sensors are usually specialized in specific applications, so they do not offer the same services. The combination of different types of sensors in a network and the use of its data requires a high-level of modularity and adaptability in the architecture.
- *Changing network topology*. Sensor networks are less stable than traditional computer networks (Yick et al., 2008). Their sensors are more prone to fail than conventional computational devices: they frequently operate unattended in environments that can lead them to malfunction, and with very limited resources. A common way to overcome sensor failure is redeploying new sensors, which further changes the network topology. These changes make that the control of the network must deal with ad-hoc topologies to attend the communications needs of a given moment with the available resources.
- *Several levels of data processing*. Processing of data happens at both local and global levels (Tubaishat, M. & Madria, S. 2003). Since sensors can be deployed over quite wide areas, the management of data may need to be contextualized, for instance to determine what

signals are relevant in a situation. Nevertheless, centralized processing is also necessary, mainly for the transformations and integration of data. Thus, the architecture of the control must deal with groups at different levels that need to coordinate.

- *Unreliable networks of reduced bandwidth.* The network established in these cases is highly unreliable when compared with wired networks (Yick et al., 2008). It is usually a wireless network where the hostile environment produces intermittent connectivity with a high variability in the conditions for communication. This issue requires solutions similar to those of the changing topology.

These challenges have been addressed in several works, though usually focusing only on the solutions for some specific issues. For instance, literature (Akyildiz et al., 2007; Baronti et al., 2007; Tubaishat, M. & Madria, S. 2003; Yick et al., 2008) reports works on routing in ad-hoc networks to minimize energy consumption, optimal data processing to reduce computation time in sensors or data integration in specific domains. However, the integration of the different solutions is not a trivial problem and research in architectures for these networks pays attention to it.

The architectures proposed for these networks usually consider some basic infrastructure and a component model. The infrastructure provides basic services for all the components, and the components model specify the interfaces and behaviour that components must provide to be integrated in the network. Examples of these works are the architectural principles in (Estrin et al., 1999) and MADSN (Qi et al., 2001), TinyOS (Hill et al., 2000) and Tenet (Gnawali et al., 2006). (Estrin et al., 1999) is probably one of the first works discussing the specific problematic of sensor networks. It advocates for architectures where data processing is performed as close as possible to the sources of those data, probably in the sensors themselves. Sensors are also responsible of communication, sometimes supported by communication specific devices. MADSN (Qi et al., 2001) proposes changing the paradigm for data integration from one where all the data are transmitted to a central processing node, to another in which mobile agents travel to the nodes that collect data and make there the processing, propagating only ellaborated data. TinyOS (Hill et al., 2000) is focused on infrastructure. It is an operating system based on micro-threading, events and a simple component model. A component has tasks, commands and handlers that are executed in the context of an encapsulated fixed-size frame, and they operate on the state of the component. Tenet (Gnawali et al., 2006) is a model of components and its supporting libraries built on top of TinyOS. It proposes a two-layered architecture with simple sensors and masters. Sensors get data and only make basic signal processing. Masters perform the integration of data using more powerful computational devices. These examples are also illustrative of the main limitations of these architectures:

- *Constraining architecture.* Most of times, the architecture includes a restricted component model. Systems need to adhere strictly to its principles, which imply developing specific interfaces and conforming to certain rules of behaviour. This is the case of TinyOS (Hill et al., 2000) and Tenet (Gnawali et al., 2006).
- *Lack of a complete vertical solution.* Although these architectures are conceived to integrate the solutions of several aspects and to give complete models on how to build a sensor network, they usually do not cover the whole design. For instance, most of the proposed examples (Estrin et al., 1999; Gnawali et al., 2006; Hill et al., 2000) are mainly related with communication and sensor management issues, but they do not say anything about the design of the sensor controllers. Even if as proposed in (Estrin et al., 1999) controllers are in sensors, the problematic of their design is focused more on

communications and data integration than on gathering data and the processing of the raw signals.

- *Lack of a supporting modelling language and development process.* The proposed architectures focus on design principles (Estrin et al., 1999; Qi et al., 2001) or infrastructure (Gnawali et al., 2006; Hill et al., 2000). Nevertheless, this is not enough to build a sensor network. Engineers need a development process that indicates them what the relevant requirements to consider are, the design alternatives, and the steps to follow in the development. The industrial use of such process demands customizable modelling languages and automated support tools.

In order to address the previous limitations, some works have proposed multi-agent systems (MAS) (Weiss, G., 2000) as the basis for the development of sensor networks. A MAS is composed of a large number of agents and other computational artefacts. These agents are social entities, which need to interact with other agents to achieve the satisfaction of system goals. Agents are goal-oriented components, i.e. they rationally choose for execution those actions that will potentially contribute to satisfy their objectives. These choices depend on the information they have in their mental states about their environment, past experiences and themselves. The works in this approach usually see sensors as devices controlled by agents. This choice meets some of the aforementioned requirements of sensor networks. Sensors are only responsible of data gathering and basic processing, while computationally expensive processes are assigned to agents. This organization gives freedom of choice to put the execution of data processing either mainly in the sensors or in the controllers. Despite of this common feature, differences between approaches are important. From the point of view of the goal of this work, they do not achieve a complete architecture and process to design it either because they are too focused on some specific issues (e.g. optimization of communications (Qi et al., 2001), only provide an architecture or just a development process which is usually a general-purpose agent-oriented software engineering (AOSE) methodology.

This work addresses these issues with a twofold solution: a standard architecture for sensor networks able to deal with different design choices; a design language oriented to the kind of abstractions appearing in it, and a development process for such systems. This chapter focuses only on the first two elements, though it gives a brief introduction to the process. To provide these elements, this work adopts as its basis a well-known general purpose AOSE methodology, INGENIAS. INGENIAS (Pavón et al., 2005) covers the full-development cycle from analysis to coding with a model-driven engineering (MDE) approach (Schmidt, D. C. 2006) supported by tools. The definition of its modelling language and tools is based on metamodels. Metamodels are a common way of defining formally modelling languages in MDE. The fact of having a formal definition of the language effectively constrains engineers during design to build proper models, and it also allows automated processes for code generation, both for tools and final systems. This allows its adaptation to new contexts by means of extensions of its metamodel.

The architecture proposed in this work considers sensor networks composed by sensors and controllers, therefore following common approaches with sensors and MAS (Pavón et al., 2007). However, it extends both definitions in several ways. A sensor is defined as an environment element with attached functioning parameters and an internal state, which can be modified using its methods. The sensor is also able to perceive events coming from its environment, and to raise events in order to notify changes in its state or that of its external environment. The architecture considers sensor networks composed by devices (which

include sensors) and actors (i.e. agents). It extends common definitions for these concepts (Pavón et al., 2007) in several ways.
A device is defined as an environment element with attached functioning parameters, an internal state, and methods to work on that state. The device is also able to raise events in order to notify changes in its state. A sensor is a device that perceives events coming from its environment. Actors are similar to controllers in other approaches, but the architecture introduces for them a neat separation of concerns with roles. A role is defined by its goals, which are related with its responsibilities, and the capabilities (i.e. tasks) and resources (i.e. devices and applications) it has to achieve them. Different role types have exclusive skills. For instance, only device controllers can communicate with devices, and the group leaders have the power to impose certain goals to the members of their groups. The current version of the architecture includes several predefined role types, but this list can be extended to address new needs of sensor networks, such as secure communications or resource assignment (Tubaishat, M. & Madria, S. 2003). These roles are played by actors, which are agents with common inherited capabilities about goal management and task execution. Their specification focuses on how they implement the specific tasks related with their roles. The architecture defines teams of roles and their interactions to perform certain tasks, for instance, the setup or the dynamic addition of sensors.
The previous definitions of sensor, roles and agent partially match those of INGENIAS external applications, roles, and agents. Nevertheless there are relevant differences. For instance, INGENIAS external applications and sensors are both environment elements characterized in terms of offered methods and produced events, but sensors extend applications considering its internal state, and how this changes as a consequence of external events and method execution. Thus, this research has modified the INGENIAS metamodel to accommodate the new concepts and being able to use the INGENIAS tools.

## 2. Important

Although there are partial solutions for the design of sensor networks, their integration relies on ad-hoc solutions requiring important development efforts. In order to provide an effective way for their integration, this chapter proposes an architecture based on the multi-agent system paradigm with a clear separation of concerns. The architecture considers sensors as devices used by an upper layer of controller agents. Agents are organized according to roles related to the different aspects to integrate, mainly sensor management, communication and data processing. This organization largely isolates and decouples the data management from the changing network, while encouraging reuse of solutions. The use of the architecture is facilitated by a specific modelling language developed through meamodelling.

## 3. Agent development with INGENIAS

INGENIAS (Pavón et al., 2005) is a MDE methodology for the development of MAS. It comprehends a specific modelling language, a software process and a support tool. Following MDE principles, it defines its design modelling language with a metamodel. This metamodel is the basis for the semi-automated development of its tool, and also guides the definition of the activities of its software process.
MAS in INGENIAS are organizations of agents, which are intentional and social entities. Agents use applications, which represent the environment and system facilities. The models

to specify these MAS describe their environment, agents and interactions, both from the static and dynamic perspectives. The modelling language also includes a simple extension mechanism for agents through inheritance relationships: a new sub-agent type inherits all the features of its super-agent type, but it can also extend or constrain them. Table 1 shows the main INGENIAS concepts used by our approach.

The support tool of the methodology is the INGENIAS Development Kit (IDK). It provides a graphical environment for the specification of MAS design models. The tool can be extended with modules. The standard distribution includes modules for documentation and code generation based on templates. A template is a text file annotated with tags. These tags indicate the places where information from models has to be injected to get a proper instantiation. The instantiated template can describe, for instance, the code for an agent in a framework, the documentation of its goals, or the configuration files for its deployment. Engineers can use code components in models to attach specific code to entities. For instance, if engineers want to generate nesC (Gay et al., 2003) code, they first need to develop a template with the general description of an agent in that language; then the code generation module reads the design models of the sensor network, and for each agent appearing in them, it generates its specific code for nesC instantiating the template (i.e. replacing the tags with information from the models that includes the code components).

| Concept | Meaning |
|---|---|
| Agent | An active element with explicit goals able to initiate actions involving other elements. |
| Role | A group of related goals and tasks. An agent playing a role adopts its goals and must be able to perform its tasks. |
| Environment application | An element of the environment. Agents/roles act on the environment using its actions and receive information from the environment through its events. |
| Internal application | A non-intentional component of the MAS. Agents/roles use it through its actions and receive information from it through its events. |
| Goal | An objective of an agent/role. Agents/roles try to satisfy their goals executing tasks. The satisfaction or failure of a goal depends on the presence or absence of some elements (i.e. frame facts and events) in the society or the environment. |
| Task | A capability of an agent/role. In order to execute a task, certain elements (i.e. frame facts and events) must be available. The execution produces/consumes some elements. |
| Interaction | A basic communication action. Agents/roles send with them information to other agents/roles. |
| Method | A basic imperative operation of an application described by its parameters and result. |
| Frame fact | An information element produced by a task, and therefore by the agents/roles. |
| Event | An information element spontaneously produced by an application. |
| Interaction | Any kind of social activity involving several agents/roles. |
| Group | A set of agents/roles that share some common goals and the applications they have access to. The behaviour of groups is specified with workflows involving its components. These workflows indicate their tasks, the elements these produce/consume and the agents/roles that carry them out. The workflows must fulfil the group goals through the achievement of the individual agent/role goals. |
| Society | A set of agents, roles, applications and groups, along with general rules that govern their behaviour. |
| Environment | The set of external applications with which the components of a MAS interact. |

Table 1. Main concepts of the modelling language of the INGENIAS methodology.

There are two main reasons for the choice of INGENIAS in this work considering available alternatives with agents (Vinyals et al., 2008). First, its modelling language is a suitable basis for the extensions required for sensor networks. It considers concepts such as agents, roles and environment applications that are required in our architecture, and covers the interactions between system components with a high-level of detail. Second, INGENIAS strictly adheres to MDE principles. It defines its modelling language with a metamodel that is also the basis of the IDK development. This facilitates the modification of the language to house additional concepts and propagating these changes to the tool. Given the complexity of the development of sensor networks (Tubaishat, M. & Madria, S. 2003; Yick et al., 2008), the availability of support tools (e.g. for coding, debugging or reporting) is mandatory to get a high productivity. Nevertheless, the IDK has the shortcoming of using an ad-hoc approach for transformations based on modules and templates, although there are ongoing efforts to support more standard approaches (García Magariño et al., 2009). The development process proposed in our work adopts standard transformation languages (Sendall et al., 2003) to manipulate models and code. This has two key advantages. First, the tools to develop and run these transformations are already externally available, so there is no need of new developments. Second, these languages focus on the description of the transformations, which facilitates their understanding as this is not blurred with low-level details about processing design models (e.g. reading the input file, managing syntax errors or generating the output file).

### 3.1 An agent-based modelling language

The design of MAS to manage sensor networks in the presented approach uses specializations of general agent concepts. The purpose of these specializations is acting as a guide for engineers: they indicate the concepts that should appear in the specifications and how they are related. The main extensions of our approach to the INGENIAS (Pavón et al., 2005) conceptual framework appear in Fig. 1. with their main relationships. The mechanism used for the metamodel extension is its direct modification (Cook, S., 2000). Note that profiles cannot be used since this is not an UML extension, and INGENIAS limits inheritance to agents.

Fig. 1. represents elements of the metamodel of the modelling language in our approach. Nodes and links respectively represent meta-entities and meta-relationships. Meta-relationships with triangles and diamonds are standard (i.e. non specific of INGENIAS) representations of inheritance and aggregation (i.e. whole-part link) relationships. Numbers in the ends of relationships are cardinality indications. The stereotypes of nodes (represented between angle brackets) are the names of the INGENIAS meta-entities that our meta-entities extend. The meta-entities have the features (e.g. attributes and relationships) of the extended meta-entities and add new features and constraints. For instance, at the meta-modelling level, there are meta-entities device and controller that are modifications of the INGENIAS meta-entities environment application and role respectively. These meta-entities are related with a meta-relationship WFUses, which is restricted to connect this pair of meta-entities. These meta-elements are instantiated in models. For example, a model can contain instances of the device meta-entity, which can only be related with instances of the controller meta-entity through instances of the WFUses meta-relationship. The rest of the section discusses the concepts present in Fig. 1.

A sensor network in the proposed architecture distinguishes between reactive and active components. Reactive components receive requests or notifications of events, and generate

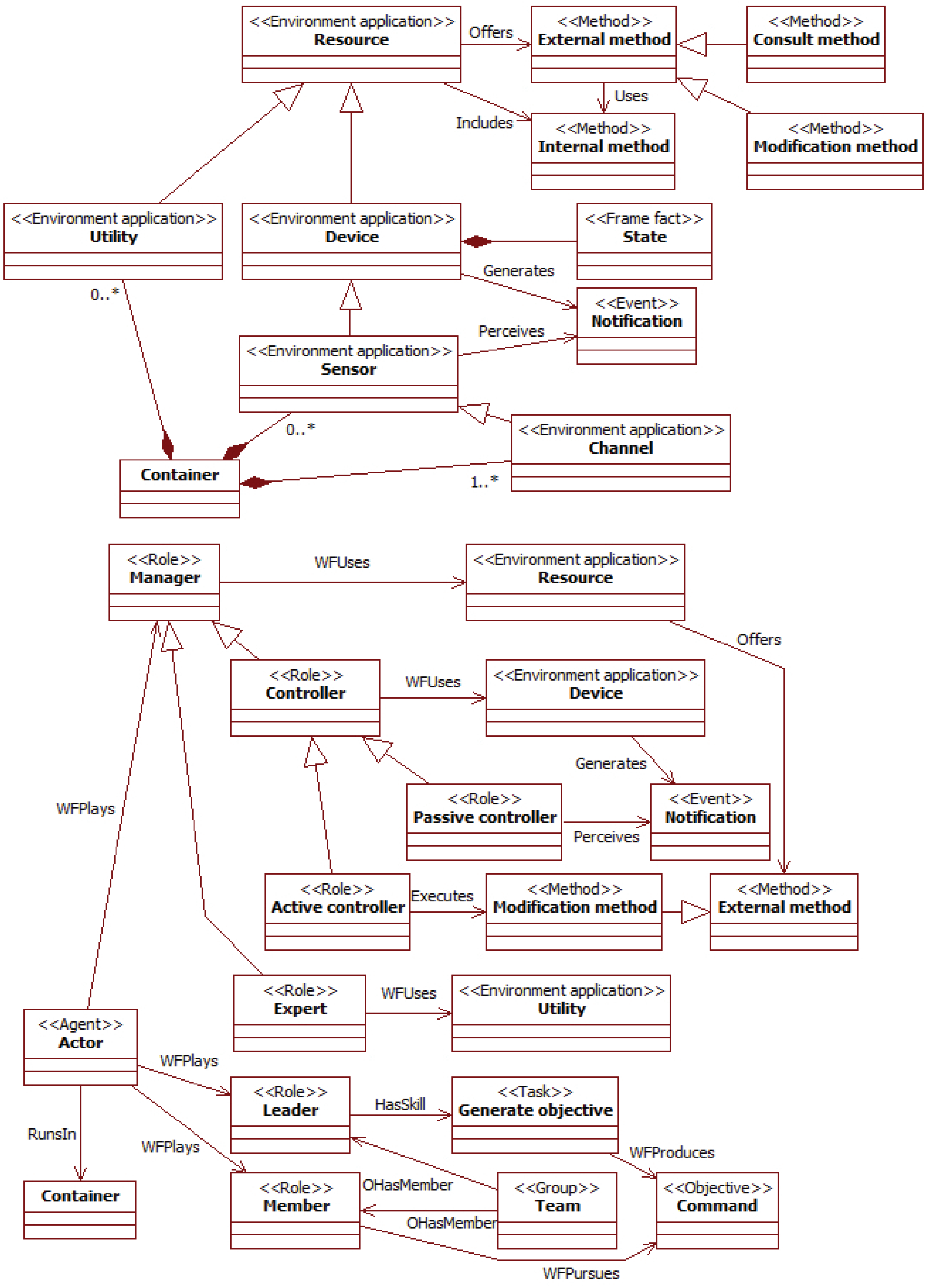
<<Environment application>> Resource
Offers
<<Method>> External method
<<Method>> Consult method
Uses
Includes
<<Method>> Internal method
<<Method>> Modification method
<<Environment application>> Utility
<<Environment application>> Device
<<Frame fact>> State
Generates
<<Event>> Notification
Perceives
<<Environment application>> Sensor
0..*
0..*
<<Environment application>> Channel
Container
1..*
<<Role>> Manager
WFUses
<<Environment application>> Resource
Offers
<<Role>> Controller
WFUses
<<Environment application>> Device
Generates
WFPlays
<<Role>> Passive controller
<<Event>> Notification
Perceives
<<Role>> Active controller
Executes
<<Method>> Modification method
<<Method>> External method
<<Role>> Expert
WFUses
<<Environment application>> Utility
<<Agent>> Actor
WFPlays
<<Role>> Leader
HasSkill
<<Task>> Generate objective
RunsIn
WFPlays
WFProduces
Container
<<Role>> Member
OHasMember
<<Group>> Team
OHasMember
<<Objective>> Command
WFPursues

Fig. 1. Partial metamodel of agent-based concepts for sensor networks.

answers for them that only depend on the input and some internal state if this exists. Active components take initiatives on their own that contribute to satisfy the system goals. The basic type of reactive component is the resource, and the actor of active component. Actors are a specific type of agents that use resources. Their work is organized through the roles they played. Roles represent prototypical aspects of the activities in the network. A role indicates the goals it pursues and the available elements to achieve them, which can be information, capabilities and resources.

A *resource* is an external application. Its specification is known, but neither its behaviour nor its interfaces can be modified. The only way to interact with it is what their external/public interfaces allow. The actions available for this external manipulation of resources are represented by *external methods*. These methods can change the internal state of the resource, i.e. *modification method*, or just consult it, i.e. *consult method. Internal methods* can be used to provide information about the internal behaviour of the resource with specification purposes, but other components of the MAS cannot invoke them. A resource may have functioning parameters that influence its behaviour. These parameters can determine for instance, the threshold of certain operations or the initially available energy. Resources represent different elements appearing in sensor networks. A *utility* is a stateless resource. It corresponds to a computational facility available for the network, such as data normalization, combination of different signals or information conversions. *Devices* are stateful resources able to generate events called *notifications*. The state is characterized in terms of frame facts, which are the units of information in INGENIAS. Devices offer specific methods to manage the subscription of other components to their notifications. A subclass of devices is *sensors*, which generate events but are also able to perceive them in the surrounding environment. Thus, the behaviour of a sensor is characterized in terms of a state machine that changes its state according to the execution of methods and the appearance of events from the environment. A *channel* is a particular type of sensor intended for communication. It is able to send and receive information over a medium and perform basic tests on it.

These resources are used by *manager* roles to provide services in sensor networks. The language distinguishes two types of managers depending on if they work with devices or utilities.

The *controller* is the role with access to devices. According to the rights it has over it, there are two types of controllers. A *passive controller* can only consult the device state with consult methods and perceive those events to which it subscribes. The *active controller* is able to make requests to change the device state using its modification methods. In this way, several access levels can be granted to controllers of the same devices.

The *expert* is the role in charge of utilities. This role specifies the knowledge and skills required to manipulate an utility, as well as how to obtain additional information that can be extracted from sequences of data manipulations over time. For instance, an expert can store information about temporal series of signals to draw conclusions about trends.

Another concept central in the proposed solution is that of *team*. A *team* is a hierarchical INGENIAS group that comprehends a *leader* role and several *member* roles. The leader has the right of posing new commands to the *members* of its team, where a *command* is a kind of objective. Roles receiving the commands must include them in their agenda, but their management of them depends on their design. The leader can be also the provider of a given service for all the members of its team. Teams facilitate setting up basic groups of collaborating roles. For instance, there are groups for the initialization of the network,

solving issues of quality of service, communications or data processing. These teams constitute design blocks that can be reused in different specifications.
The previous roles are played by roles and actors. When a role plays another role, it adds the features of that role to its own ones. The actors are agents with common skills for the management of goals (e.g. decomposition, checking their state or removing when satisfied), planning for their achievement (in terms of the available information, resources and capabilities) and basic communications (both with agents and resources). When an actor plays a role, it fulfils the standard behaviours specified by the role, that is, it implements its capabilities, has actual access to its resources, and manipulates the related goals and information. The actor can have additional elements beyond those of its roles. Note that an actor manipulates all these elements globally. For instance, the satisfaction of a goal linked to a certain role can be the result of the information produced with a capability related with another role.
The previous elements run in *containers*, which represent deployable computational devices. A container has basic processing capabilities that allow the execution of agents, and at least one channel for communication. Additionally, it can include an arbitrary number of resources. Note that, given the relationships and constraints in the metamodel, a device and its managers run in the same container.
In order to provide a simple extension mechanism for the language, this approach also generalizes the INGENIAS inheritance relationship. It is not constrained to agents but can be applied to any concept with an equivalent meaning: a sub-concept inheriting from a super-concept has all its features but it can extend or constrain them with additional models.

## 4. Architecture for sensor networks

The metamodel for sensor networks just defines the modelling primitives that can be used when specifying these networks as MAS. However, it cannot specify how these elements should interact to provide the expected functionality. The architecture provides this information. This section focuses on its description through its main teams. The list is not exhaustive, as more teams can be specified to address new needs. The description of teams includes their purpose, and the characterization of their leader and member roles regarding their responsibilities. Note that when talking about roles performing actions, it is really the actors playing those roles that perform the actions, as roles are just functional abstractions.
The *initialization* team is aimed at setting up all the components of a container and providing them with the initial information required for their proper functioning. Its team leader is the initializer and its members play the role of targets of the initialization. The *initializer* creates all the actors in its container and sends them the information about the managers they play. Then, each manager receives a list of the assigned resources, and the notifications and external methods it can use. If required, it can also obtain information to initialize the resources. Besides, each manager receives information about all the teams it belongs to, including the type of team, its leader and the role of that manager in it. Note that these teams can involve roles whose actors are not running in the same container.
An *information process* team focuses on the generation of information from the data of devices. Its team leader is a consumer for that information. It organizes the gathering and processing of data. Team members play the role of *providers* of information and can be passive controllers or experts. The activity of these teams can begin either with a request from the customer or with a notification from a device. In the second case, a passive

controller provider captures an event raised by a device and notifies it to its consumer. From this point forward, both scenarios are the same. The consumer may send additional requests to its manager providers: to passive controllers in order to collect additional data; to experts to further manipulate these data before their use. Note that with this approach, the consumer itself can be regarded as a manager that provides services of a higher-level, as it encapsulates the interactions with a group of resources and its managers.

*Communication* teams refine the INGENIAS communication schema, as they give further details about how interactions are transmitted between different actors and roles. They manage communication through *channels*. The *communicator* is both the team leader and the active controller of the channel. The rest of the members of the team play the role *customers* in the communication. All the customers in a communication team are able of direct communication between them, but they need to ask its communicator for external communication outside the team. These teams encapsulate the use of the communication infrastructure and related algorithms, which makes transparent the communication capabilities of other network elements from the design point of view. Engineers only need to guarantee that each role or actor that needs to communicate belongs to a communication team in order to have access to a communicator. In order to optimize communications (e.g. latency or energy consumption) and perform message routing, communicators need to build a rough map of the nearby communicators. Containers have a limited range of communication, so some messages may need several hops to reach its final destination. To build the map, a communicator broadcasts a request of information amongst other communicators in range. Available communicators answer this request with information about the features of their service, and take note of the sending communicator.

*Load balancing* teams are intended to keep the quality of service in the network. Sensor networks face to several situations that can require their dynamic reconfiguration. Some of them were outlined in the introduction, such as failure of sensors or communications, but also sensors overloaded with requests or replacement of the failing customer for some data. Although different, all these situations are solved through the collaboration of two sub-teams. First, there is *a failure notification* team where a team leader *referee* controls a group of team member *watchers* that can warn of potential failures in the behaviour of some observed elements of the network. For instance, a controller can be the watcher of a sensor: when this sensor depletes its energy, it does not longer answer the requests of its watcher, which raises to its referee the information about the failure. The referee evaluates that information and if it determines that there is need of acting, a repairer team begins working. A *repairer* team has as leader a *dispatcher* governing a set of *referees* and *initializers*. When a dispatcher receives the notification of a failure, it looks for some replacement. The replacement can be obtained either asking other referees in the team for a component with similar features or asking an initializer to create a new one if possible. For instance, in the case of failure of a sensor, the replacement could be another sensor in a container near the location of the original one, but if an expert is failing, a new one can be created and assigned to the utilities of the original one. The dispatcher informs of the replacement to the involved referees, which send to the initializers in their containers the information to update. For instance, adjustments need to be made in the state of the replacement or the teams depending on it.

Note that any container must have running at least two teams. The initial setup requires one initialization team, and integration with other elements of the network a communication team. Executing these teams requires at least one actor which plays the initializer and communicator roles.

The architecture involving these teams pursues satisfying three main objectives. First, it facilitates the design of sensor networks decoupling the different responsibilities in roles and teams. Second, it looks for networks that can semi-autonomously reconfigure themselves to address new situations, a concept present in current research in autonomic computing (Kephart, J. O. & Chess, D. M., 2000). Third, it achieves the extensibility of the design of systems to control sensor networks through new teams.

## 5. Development process

In this chapter a simple model-driven development process customized is included to develop the control system of sensor networks following the architecture in section 4. A model-driven process focuses the development on design models. Engineers refine these models from abstract representations to those models closer to the intended target platform, and finally to code. The refinements are partially supported by automated transformations. The process proposed in this approach is based on the software process of the INGENIAS methodology (Pavón et al., 2005). It adds to the INGENIAS process several specific activities aimed at identifying the elements required in a sensor network. These elements are those defined in the modelling language and organized in the teams of the architecture. Fig. 2. shows the resulting process. Activities 1-7 are specific of the current approach, while activities 8-13 summarize INGENIAS activities. The process takes as input a previous analysis of the data required as output of the network and the sensors able to provide them, and produces as output the code of the control system for the sensor network.
The design of the network begins with activity 1. Engineers determine the containers of the network, i.e. the computational devices able to execute code and transmit information. These are usually the sensors, but also additional devices such as computers or communication facilities can be considered here. This activity also identifies the resources: the sensors that gather data from the environment; the utilities that represent services that actors use to process data. The activity distinguishes the two aspects of the sensor, as resource and container. Note that the modelling language provides different concepts for these aspects, and therefore assign to them particular features that must be considered in the design models. When these elements have been identified, engineers assign them initialization and communication teams. As discussed in section 4, these teams are mandatory for every container.
Decision 2 and activities 3-4 are intended to organize complex processing and integration of data. According to the architecture, information process teams are responsible of these activities. Engineers identify in decision 2 and activity 3 specific data that must be generated in the network. For each group of data, activity 4 designs the corresponding team. First, engineers discover the sensors that provide the source data. For each sensor, they must assign at least one active controller and a passive one. The first one is required in the initialization, and the second one to provide access to sensor data. Next, engineers must identify the data transformations required to get the final information. Some of them are achieved using utilities of the network. For these utilities, engineers assign an expert. Finally, the team is composed by the passive controllers of the sensors and the experts of the utilities playing the role of providers, and a customer to integrate and consume the information. The identification of this kind of teams finishes when all the complex calculation of data has a team assigned.

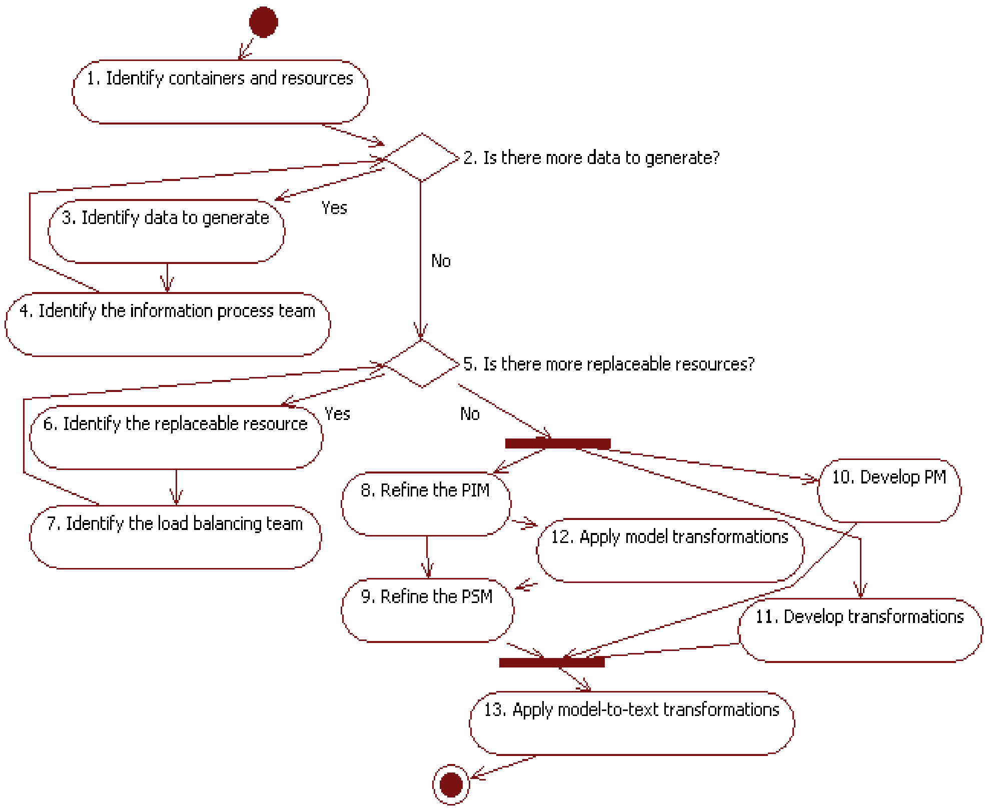
1. Identify containers and resources
2. Is there more data to generate?
Yes
No
3. Identify data to generate
4. Identify the information process team
5. Is there more replaceable resources?
Yes
No
6. Identify the replaceable resource
7. Identify the load balancing team
8. Refine the PIM
9. Refine the PSM
10. Develop PM
11. Develop transformations
12. Apply model transformations
13. Apply model-to-text transformations

Fig. 2. Process for the development of sensor networks.

Decision 5 and activities 6-7 are intended to specify the teams that manage the dynamic adaptation of the network. Engineers begin this design with decision 5, where they find out what the elements are that can fail or be incrementally set up or deployed during the working of the network. This identification considers resources, roles and actors. For every element identified in this decision, activity 6 carries out an analysis regarding its potential replacements, and how they can be located and evaluated to find the best suited if several are available. Activity 7 designs the specific team related with this replacement. It includes a watcher that monitors the element. In the case of a device it is a passive controller, for a utility it is an expert, and for a role or agent it can be a customer that communicates with it. The team also needs a referee that evaluates when the failure needs to be notified for a potential replacement. The repairer team includes referees related with the same type of elements and similar features. For instance, for sensors they can be referees of nearby sensors and for roles other agents in the same container able to work with the same resources. As an alternative, initializers can be used to set up new roles or agents in these teams. Each of these teams must also identify its dispatcher, which selects the best alternative for a required replacement.

After these activities, engineers have available PIM of the resulting system according to the architecture. These PIM describe the devices, agents and roles, the information they exchange, and their interactions; they do not contain details on the final target platform, for instance about energy levels or low-level control commands for the sensors. Activities 8-13 follow the INGENIAS process to refine these models and generate the final code of the control system.

Activity 8 adds several INGENIAS PIM to the MAS specifications. Organization models define agents and groups outside the architecture, and assign to the groups workflows that describe their work. This allows refining the teams when complex processing of data needs further specification. Agent models refine actors and roles with additional goals, capabilities and information. These models also establish the pieces of information whose appearance determine when a goal is satisfied or failed. Tasks/Goals models map tasks with the goals that satisfy them, and hierarchically decompose goals and tasks into sub-elements. Interaction models describe actor interactions in terms of goals pursued, information exchanged and tasks performed. These models provide the details of the previous architectural design, though they are not always required. For instance, if engineers do not need to refine teams beyond what is said in the architecture, they do not use organization models.

Activity 9 and 10 develop the models required for the final target platform. Activity 10 develops the PM corresponding to the target platform. These PM include information about how to translate general concepts to specific elements in the platform. As explained in section 3, INGENIAS uses templates to represents PM. In case that these PM are available from previous projects, activity 10 can be omitted. Activity 9 develops the PSM of a specific design for the target platform. The PSM provide two main types of information. First, resources include their functioning parameters for the target platform, which can describe their limits about energy, memory or computational power. Second, engineers provide with code components attached to modelling entities the code specific for them. That is, part of the code required for the final system cannot be extracted from models, as models abstract the specific low-level details of the behaviour of systems. For instance, there are not modelling primitives to describe complex algorithms, and templates only contain general code for concept types in a platform. Engineers can include the remaining information attaching INGENIAS code components to the elements in models.

Activity 11 considers the development of the transformations that support the semi-automated refinement of PIM to PSM in activity 12, and the generation of code from PSM in activity 13. In the case of an INGENIAS development, transformations are implemented as IDK modules. These modules support model transformations and model-to-text transformations. Model transformations are useful to represents standard refinements of model concepts. For instance, each actor needs several goals to manage its planning cycles (e.g. collect information, discard non-achievable goals, look for achievable goals), but these are standard and engineers do not need to write them for each actor; a transformation can automatically generate these goals for the available actors. The best-known example of model-to-text transformation is code generation. In this case, the IDK includes a module for this purpose in its standard distribution. For a given specification and target platform, this module operates as follows: (1) it identifies the templates for the concepts present in the specifications and the target platform; (2) it traverses the templates looking for their tags; (3) when it found a tag, it replaces the tag with information from the models, which can be the content of a code template; (4) it returns the instantiated template as its output, which is the code of the concepts. In this way, changing the target platform for a given design only

requires using different PM (i.e. code templates) and changing the attached code components.
Note that though Fig. 2. shows a sequence of activities, a true development needs to carry out several iterations of these activities. For instance, engineers can discover when they are developing their PSM in activity 9 that some teams are missed, and they will need to return to activities 2-7. Moreover, activities 1-7 need further refinement to provide more guidance depending on specific application contexts.

## 6. Related work

This section compares the proposed approach with related work in sensor networks and MAS. The introduction already discussed different perspectives on the design of sensor networks. This section follows this classification and distinguishes between integral solutions with architectures and partial solutions for specific aspects. Among architectures, there are examples focused on the infrastructure and others on the high-level design of the network. Transversal to these approaches, some researchers have proposed the use of MAS for the development of the related control systems.
Architectures for sensor networks focused on infrastructure provide a platform with basic services for the sensor network. This platform has a component model that those elements to integrate in the network must fulfil. In this group can be included operating systems (e.g. TinyOS (Hill et al., 2000) and Contiki (Dunkels et al., 2000)), programming languages (e.g. nesC (Gay et al., 2003)) and middleware (e.g. MORE (Wolff et al., 2007), RUNES (Costa et al., 2005), SMEPP (Caro-Benito et al., 2009) and Tenet (Gnawali et al., 2006)). These works and ours appear at different levels of abstraction when considering the development of sensor networks and their control systems. According to MDA (Kleppe et al., 2003), the models based on the architecture proposed in this work are PIM that use highly abstract primitives to model sensor networks. These abstract elements are mapped to the constructions available in these implementation platforms. For instance, the concept of team in our architecture can be partially supported in SMEPP (Caro-Benito et al., 2009) with the concept of group, which provides mechanisms for authentication and authorization, communication between agents can be implemented with μSOA messages of MORE (Wolff et al., 2007). The information of these platforms would appear in our approach as PM. Engineers would refine the PIM of our architecture in that provides the information for that specific implementation. This refinement would be partly implemented with automated transformations from PIM to PSM, for instance to create the structure of μSOA messages, and partly manual, for instance the actual content of messages. If required, abstract components of these architectures could appear in the architecture of this work as additional roles and teams. The code generation module of the IDK would generate the code for the control system from the final PSM and PM.
Architectures considering the high-level design of the network have adopted usually the form of guidelines. Either they just give some abstract design principles or they consider also a development process. From the point of view of the design principles, the flexibility of the proposed architecture allows it adopting the principles underlying a variety of these approaches. For instance, carrying out the processing of data as close as possible to their sources (as (Qi et al., 2001) recommends) means that the actors playing the roles of information process teams should run in the same container, and moving that processing to more powerful computational devices (as proposed in (Gnawali et al., 2006)) splits these

actors in different containers. In both cases the design of roles and teams is the same, and only the initialization information actually changes. The proposed architecture is not intended however for mobile agents as those in (Qi et al., 2001; Tong et al., 2003). Actors in the proposed architecture are not able of redeploying in a container different from that where the initializers create them. However, the initializers could be modified to allow this kind of behaviour. It would be enough to allow initializers to collect information about the actor that wants to migrate (e.g. current state, teams or available resources), and send it to the target container where another initializer would use it to create another actor with the same data. Of course, this migration would also demand checking that the resources and managers that the actor needs are available in the target container.

This section has already mentioned works based on agents (Botti et al., 1999; Hla et al., 2008; Jamont, J. P. & Occello, M., 2007; Pavón et al., 2007; Qi et al., 2001), but some of them deserve further discussion given the similarities with our work. (Hla et al., 2008) establishes some guides for the design of MAS for sensor networks and uses some concepts common with our approach, as controllers, sensors and providers. They also consider concepts that our approach can incorporate, such as directory facilitators to refine the location of sensors with certain features. However, these roles are informally defined in terms of their responsibilities and the set is closed. In this sense, our approach with a specific modelling language and the possibility of defining teams facilitates customization. Besides, (Hla et al., 2008) does not consider a development process for control systems. (Botti et al., 1999; Jamont, J. P. & Occello, M., 2007; Pavón et al., 2007) present development process for control systems. ARTIS (Botti et al., 1999) is a methodology for holonic manufacturing systems that includes the use of sensors. It considers aspects of real time, but ignores issues such as limited resources. (Jamont, J. P. & Occello, M., 2007) is tailored for sensor networks and has been validated with real projects. Though it considers automated generation of code, it does not offer a standard process for it, as our approach does with MDE. This makes more difficult reusing available infrastructure for development and reusing the design models of previous projects. (Pavón et al., 2007) deserves special mention as it also considers INGENIAS for the design of sensor networks. As a matter of fact, both approaches represent complementary points of view. The approach proposed in this work extends the modelling language of INGENIAS with new concepts, and establishes patterns and guidelines to address the design of these networks with its architecture. These tasks correspond to activities 1-7 in Fig. 2. Since these models are INGENIAS models, their refinement to the running code can follow any suitable INGENIAS development process. These tasks correspond to activities 8-13 in Fig. 2. In particular, this refinement can follow (Pavón et al., 2007), which is targeted for sensor networks. Thus, these works can be seen as part of an ongoing effort to provide engineers with a tailored methodology and development process for sensor networks.

## 7. Conclusions

The presented approach is intended to facilitate the high-level design of sensor networks based on MAS. It includes an agent-oriented modelling language with specific extensions and an architecture describing how these elements interact to achieve the standard functionalities of these networks.

The modelling language is built around three main concepts. Resources are the passive elements in the network. They are modelled in terms of their available methods. Their subtypes include sensors and data processing utilities. Sensors add to resources a state and work with events, both perceived from their environment and raised to inform to their controllers. These abstractions cover the most common uses of sensors in previous works. The active elements of the network are designed as roles. Roles are common abstraction in MAS defined in terms of their goals, capabilities to achieve them, and their resources and information. Managers are in our approach the roles governing resources. They can have different access rights in order to organize the use of the resources. The final element of the language is teams, which are hierarchical groups of roles aimed at performing some collaborative activities in the network. Actors running in containers implement the roles.

The architecture works with these concepts to specify teams that define standard aspects of behaviour in these networks. It identifies teams for the initialization or redeployment of containers, the management of data (including collection, processing and integration), communications and load balancing (or adaptation of the network to changes in the environment or its elements).

The proposed solution is intended to be flexible in several ways. First, it allows accommodating new or modified concepts for specific needs through changes in its metamodel. Using model-driven techniques, engineers propagate these changes to the supporting tools. Second, the specialization of concepts with inheritance relationships and the organization of systems around teams cover a variety of approaches, so it allows incorporating existing research in the area. Third, the use of a MDE approach facilitates reusing the knowledge present in the definition of teams. These teams can become the basic building blocks for sensor networks with MAS, as their models can incorporate information for the final code generation. Only models and transformations related with the control of specific sensors and particular manipulations of data need to be replaced in the system. If new teams were required, they could be modelled as extensions of concepts presents in the architecture as done with standard teams.

The main concern in the application of the proposed approach is the difficulty to model the low-level details of sensor networks, such as energy consumption of routing algorithms for messages. At the moment, the only mean to do that is attaching code snippets to entities in design models for the code generation. There are plans to extend the modelling language with additional primitives to describe some low level issues. For instance, methods can be modelled with additional state machines, and certain standard data transformations can be added as instances of the methods of utilities. Moreover, this work has applied the standard INGENIAS development process for part of its process.

## 8. References

Akyildiz, I. F., Melodia, T., Chowdhury, K. R. (2007) A survey on wireless multimedia sensor networks. Computer Networks 51(4), pp. 921-960.

Baronti, P., Pillai, P., Chook, V. W. C., Chessa, S., Gotta, A., Hu, Y. F. (2007) Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and ZigBee standards. Computer Communications 30(7), pp. 1655-1695.

Botti, V., Carrascosa, C., Julián, V., Soler, J. (1999) Modelling agents in hard real-time environments. In Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99), Lecture Notes in Computer Science 1647, pp. 63-76.

Budinsky, F. (2003) Eclipse Modelling Framework: Developer's Guide. Addison Wesley.

Caro-Benito, R. J., Garrido-Márquez, D., Plaza-Tron, P., Román-Castro, R., Sanz-Martín, N., Serrano-Martín, J. L. (2009) SMEPP: A Secure Middleware For Embedded P2P. In Proceedings of the 2009 ICT Mobile Summit, pp. 1-8.

Casbeer, D. W., Kingston, D. B., Beard, R. W., McLain, T. W. (2006) Cooperative forest fire surveillance using a team of small unmanned air vehicles. International Journal of Systems Science 37(6), pp. 351-360.

Cook, S. (2000) The UML family: Profiles, prefaces and packages. In Proceedings of the 3rd International Conference «UML» – The Unified Modelling Language (UML 2000), Lecture Notes in Computer Science 1939, pp. 255-264.

Costa, P., Coulson, G., Mascolo, C., Picco, G. P., Zachariadis, S. (2005) The RUNES middleware: A reconfigurable component-based approach to networked embedded systems. In Proceedings of the 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC'05), vol. 2, pp. 806-810.

Dunkels, A., Gronvall, B., Voigt, T. (2004) Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN 2004), pp. 452-462.

Estrin, D., Govindan, R., Heidemann, J., Kumar, S. (1999) Next century challenges: Scalable coordination in sensor networks. In Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM 1999), pp. 263-270.

Gay, D., Levis, P., Von Behren, R., Welsh, M., Brewer, E., Culler, D. (2003) The nesC language: A holistic approach to networked embedded systems. In Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI 2003), pp. 1-11.

Gnawali, O., Jang, K. Y., Paek, J., Vieira, M., Govindan, R., Greenstein, B., Joki, A., Estrin, D., Kohler, E. (2006) The Tenet architecture for tiered sensor networks. In Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys 2006), ACM Press, pp. 153-166.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K. (2000) System architecture directions for networked sensors. ACM Sigplan Notices 35(11), pp. 93-104.

Hla, K. H. S., Choi, Y., Park, J. S. (2008) The multi agent system solutions for wireless sensor network applications. In Proceedings of the 2nd KES Symposium on Agent and Multi-Agent Systems – Technologies and Applications (KES-AMSTA 2008), Lecture Notes in Computer Science 4953, pp. 454-463.

Jamont, J. P., Occello, M. (2007) Designing embedded collective systems: The DIAMOND multiagent method. In Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), pp. 91-94.

Kephart, J. O., Chess, D. M. (2003) The Vision of Autonomic Computing. Computer 36(1), pp. 41-50.

Kleppe, A. G., Warmer, J., Bast, B. (2003) MDA Explained: The Model Driven Architecture - Practice and Promise. Addison-Wesley.

García-Magariño, I., Rougemaille, S., Fuentes-Fernández, R., Migeon, F., Gleizes, M. P., Gómez-Sanz, J. J. (2009) A Tool for Generating Model Transformations By-Example in Multi-Agent Systems. In Proceedings of the 7th International Conference on

Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009), Advances in Soft Computing 55, pp. 70-79.

Lorincz, K., Malan, D. J., Fulford-Jones, T. R. F., Nawoj, A., Clavel, A., Shnayder, V., Mainland, G., Welsh, M., Moulton, S. (2004) Sensor Networks for Emergency Response: Challenges and Opportunities. IEEE Pervasive Computing 3(4), pp. 16-23.

Object Management Group (OMG) (2006) Meta Object Facility (MOF), Core Specification, Version 2.0. January 2006, http://www.omg.org

Object Management Group (OMG) (2009) OMG Unified Modeling Language (OMG UML), Superstructure, V2.2. February 2009, http://www.omg.org

Pavón, J., Gómez-Sanz, J., Fernández-Caballero, A., Valencia-Jiménez, J. J. (2007) Development of intelligent multisensor surveillance systems with agents. Robotics and Autonomous Systems 55(12), pp. 892-903.

Pavón, J., Gómez-Sanz, J. J., Fuentes, R. (2005). The INGENIAS Methodology and Tools. In Henderson-Sellers, B., Giorgini, P. (eds.). Agent-Oriented Methodologies, Idea Group Publishing, chapter IX, pp. 236–276.

Qi, H., Iyengar, S. S., Chakrabarty, K. (2001) Multiresolution data integration using mobile agents in distributed sensor networks. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 31(3), pp. 383-391.

Schmidt, D. C. (2006) Model-Driven Engineering. IEEE Computer 39(2), pp. 25-31.

Sendall, S., Kozaczynski, W. (2003) Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Software 20(5), pp. 42-45.

Son, B., Her, Y., Kim, J. G. (2006) A design and implementation of forest-fires surveillance system based on wireless sensor networks for South Korea mountains. International Journal of Computer Science and Network Security 6(9B), pp. 124-130.

Tong, L., Zhao, Q., Adireddy, S. (2003) Sensor networks with mobile agents. In Proceedings of the IEEE Military Communications Conference (MILCOM 2003), vol. 1, pp. 688-693.

Tubaishat, M., Madria, S. (2003) Sensor networks: an overview. IEEE Potentials 22(2), pp. 20-23.

Vinyals, M., Rodriguez-Aguilar, J. A., Cerquides, J. (2008) A survey on sensor networks from a multi-agent perspective. In Proceedings of the 2nd International Workshop on Agent Technology for Sensor Networks (ATSN 2008), pp. 1-8.

Weiss, G. (ed.) (2000) Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press.

Wolff, A., Michaelis, S., Schmutzler, J., Wietfeld, C. (2007) Network-centric Middleware for Service Oriented Architectures across Heterogeneous Embedded Systems. In Proceedings of the 11th International IEEE Enterprise Distributed Object Computing Conference (EDOC 2007), Middleware for Web Services Workshop, pp. 105-108.

Yick, J., Mukherjee, B., Ghosal, D. (2008) Wireless sensor network survey. Computer Networks 52(12), pp. 2292-2330.