IDK and ICARO: developing Multi-agent Systems in support of Ambient Intelligence

José M. Gascueña [a], Elena Navarro [a,b], Antonio Fernández-Caballero [a,b], Patricia Fernández-Sotos [c] and Juan Pavón [d]

[a] *Instituto de Investigación en Informática de Albacete (I3A), 02071-Albacete, Spain*
[b] *Universidad de Castilla-La Mancha, Departamento de Sistemas Informáticos, 02071-Albacete, Spain*
E-mail: {Elena.Navarro, Antonio.Fdez}@uclm.es
[c] *Universidad de Castilla-La Mancha, Facultad de Medicina, 02071-Albacete, Spain*
[d] *Universidad Complutense de Madrid, Facultad de Informática, Departamento de Ingeniería del Software e Inteligencia Artificial, c/ Profesor José García Santesmases, s/n, 28040-Madrid, Spain*

AbstractAn important component of Ambient Intelligence (AmI) is the massive deployment of a new generation of intelligent agents, embedded in the environment and adaptive to the users' profiles, preferences and needs. That is why, the use of agent-oriented methodologies and frameworks is almost mandatory to easily develop software for AmI scenarios. ICARO is a software framework that promotes the use of different organizational and behavioral patterns to implement multi-agent systems (MAS). Its extensive use in several projects demonstrates a substantial increase in software productivity. Also, in order to reduce the coding effort it is usual to design MAS at a higher level. In this sense, code generation from MAS specifications into ICARO framework has been performed. INGENIAS Development Kit (IDK) supports both the specification of MAS models, including any feature required to implement MAS with ICARO, and a set of facilities for code generation. This paper describes the development of AmI applications thanks to the integration of ICARO and IDK. Two IDK modules have been developed, namely a "code generator" and "code update" module.

Keywords: Multi-agent systems, Agent-oriented software engineering, Code generation, Ambient Intelligence, Assisted living systems, Ambient assisted systems, Smart homes.

1. Introduction

Ambient Intelligence (AmI) is being a very active topic of research in the last decade. Ducatel et al. [14] envisioned some of the ideal scenarios that AmI should be able to offer to the society, community as well as individuals. As stated by the authors, AmI promotes the development of innovative and intelligent user interfaces "embedded in an environment that is capable of recognizing and responding to the presence of different individuals in a seamless, unobtrusive and often invisible way". AmI user interfaces become transparent (people do not perceive complexity neither presence) and they are "intelligent" to react in a proactive and sensitive way [1] at the same time. The idea of making technology transparent but usable to people has opened the door to its exploitation in a wide variety of previously unforeseen systems. For instance, it is being widely exploited in the context of *smart homes*, also called *ambient assisted systems* and *assisted living systems*. They are home-like environments that exploit AmI to respond to the behavior of their residents and to provide them with novel services and facilities [13]. Home care is one major category of smart home application. Some systems have been developed for detecting falls of elders by processing the information of sensor infrastructures [37]. Other interesting systems are being applied in the context of health-care to detect problems such as heart breaks by processing context information [11]. These systems target at elders to provide them with independence and quality of life.

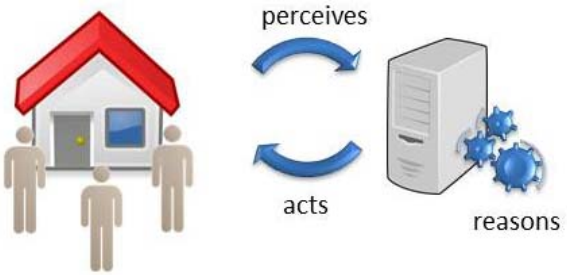

Figure 1. AmI system interacting with its environment.

AmI systems take advantage of Artificial Intelligence (AI) techniques as the means to resemble humans when interacting with their environment. For this reason, an AmI system *perceives* the state of the users and/or the environment, *reasons* about the collected data by using AI techniques such as intelligent agents, and *acts* accordingly to achieve the expected goals (see Fig. 1). Moreover, agents are a good in modeling, simulating, and representing meaningful entities such as rooms and persons in AmI systems [34]. The AmI systems properties turn their development into a challenging work demanding adequate tools that assist throughout a well established software development process. Both tools and process should be able to adapt to the diversity of AmI devices and communication technologies from an implementation point of view. The major contribution of this paper is to introduce our experience after developing couple of software modules facing the mentioned issues. The software consists in a code generator module and module to support code update. The modules have not been developed from scratch but INGENIAS Development Kit (IDK) [20], the tool supporting the agent-oriented software engineering methodology named INGENIAS [31], has been used. IDK provides a template-based proprietary mechanism for developing new modules which automatically generate code for any target language. As far as we know this functionality is not provided by any other tool dedicated to designing multi-agent systems (MAS) (e.g. Prometheus Design Tool (PDT) [30], Tool for Agent Oriented visual Modeling for the Eclipse platform (Taom4E) [26] or DSML4MAS development environment (DDE) [38]).

The two aforementioned modules generate code for ICARO framework that provides high-level software components to facilitate the development agent-based applications. Moreover, it promotes the use of different organizational and behavioral patterns that enable the specification of MAS at a higher level during design phases. With this aim in mind, ICARO provides engineers with concepts and models, together with a customizable MAS design, Java code fully compatible with software engineering standardsas Also, it can be integrated into the most popular integrated development environments (IDE). Moreover, it is independent of the agent architecture, this way enabling developers to create new architectures and integrate them into the framework. This is a clear difference with regard to other agent frameworks, such as JACK [40] or JADE [4], as they provide a middleware instead of an extensible architecture to establish the communications among agents. This advantage provides developers with the necessary flexibility to deal with the diversity of AmI devices and communication technologies. An additional advantage is that the framework already implements functionality for automatic component management, application initialization and shutdown. This reduces the developers' workload and ensures that all components are under control. This last functionality is not usually provided by other frameworks.

The rest of the article is organized as follows. Section 2 describes the related work. Then, section 3 introduces IDK modules developed for ICARO. After that, a case study illustrates our approach in section 4. Finally, some conclusions are provided in section 5.

2. Related work

As stated by Cook et al. [10] AmI technologies are expected to be sensitive, responsive, adaptive, transparent, ubiquitous, and intelligent. The three first features greatly rely on the *context-aware computing* field, but transparency and ubiquity depend on the area of *ubiquitous computing*, also called *pervasive computing*. But it is intelligence which becomes a critical feature as it makes AmI systems more sensitive, responsive, adaptive, transparent and ubiquitous. The main reason is that the intelligence feature helps in understanding user environments and, consequently, in providing them adaptive assistance [9]. This explains why AmI entails contributions from different AI areas, such as machine learning [3], ontologies [24], neural networks [9] and, specially, multi-agents systems (MAS).

Indeed, MAS are especially good at modeling real-world and social systems, where problems are solved in a concurrent and cooperative way without the need of reaching optimal solutions [34]. Indeed, AmI

proposes the development of context aware systems equipped with devices that recognize context and act accordingly. Agents provide an effective way to develop such systems since agents are reactive, proactive and exhibit an intelligent and autonomous behavior [2]. Agents react to humans based on information obtained by sensors and their knowledge about human behaviors within AmI applications [6].

As aforementioned, the development of AmI systems is not a trivial issue, being necessary the use of tools and well-defined processes that guide the stakeholders. Taken into account the relevance that MAS has to AmI, several alternatives providing the necessary support have recently been developed. One of them is the methodology defined by Serrano et al. [36], based on the use of MAS-oriented simulations for the validation of AmI-based ubiquitous computing systems. This proposal is very interesting as it helps to validate AmI applications whose real tests would be impractical because of the unavailability of resources, high cost, and so on. Unfortunately, the methodology only provides guidelines for validation purposes and not for the whole development of AmI systems; so it has to be used jointly with other proposals. Muñoz et al. [28] present an argumentative multi-agent architecture that exploits semantic web ontologies to discover inconsistent contexts, but the approach does not offer a complete support for the development of AmI systems from an implementation point of view. Yi-bin et al. [41] have also defined a set of models applicable for design and development that unfortunately cannot be extended to provide the necessary flexibility. Ayala et al. [2] have recently developed a platform-neutral framework called MalacaTiny. It is a quite powerful framework which manages the diversity of communication technologies by using aspect-oriented programming. However, it has been specifically designed for mobile applications so that its capabilities to develop general AmI systems remain unexplored. Other framework is JaCaMo [7], which encompasses a multi-dimensional concept (organization, agent and environment) to develop MAS. Unfortunately, the JaCaMo related agent programming language uses BDI [35] concepts, reducing its reusability in different applications. Although these proposals are relevant to the development of AmI systems and provide great advantages, all of them experience some serious problems when they have to deal with two challenging issues at the implementation level, namely the diversity of AmI devices and the variety of communication technologies.

A clear alternative is ICARO which provides patterns to build reactive [18] and cognitive [22] agents. The use of component patterns for modeling MAS is a differentiating factor from other agent-oriented programming frameworks and languages [5]. Moreover, while other agent-based platforms that are FIPA-compliant (e.g. JADE) focus mainly on communication standards, ICARO targets on providing high-level software components for easy development of complex agent behavior, agent coordination, and MAS organization. Regrettably, to date there are no tools for modeling MAS applications and generating code for this framework. So, there is a need for code generation from design specifications to ICARO framework.

Code generators are useful tools for software development due to the evident benefits provided by their exploitation [29], [15]. One of these benefits is an improved productivity since the time necessary to perform coding tasks is reduced. Another important benefit is that the quality of the developed systems is also improved, as the generated code (usually) does not have bugs. MAS are not an exception to this rule. Several tools for developing MAS applications [33] already provide functionality to generate code for a given agent programming language or framework. For example, the supporting tool of the Prometheus methodology offers a code generation facility to automatically produce JACK agent language code [40]. Taom4E [27], a tool for the development of software following the TROPOS methodology [26], includes functionality to generate code for Jadex language [8]. DDE [38] is an environment for the development of MAS that is based on a Domain Specific Modeling Language for MAS and supports code generation for JACK and JADE languages [4]. Another proposal develops a code generator with MOFScript (see http://marketplace.eclipse.org/content/mofscript-model-transformation-tool) that transforms agent models, compliant to its meta-model PIM4Agents, to code for the MalacaTiny platform-neutral framework [2].

Finally, our code generator was developed with IDK [20]. Several reasons led us to this decision. First, it offers a graphical editor for modeling MAS applications and functionality for developing new modules able to automatically generate code for any target language. This graphical editor, generated from the INGENIAS meta-model [31], can be extended introducing new concepts and relations needed to build new MAS applications. Afterwards, the IDK is regenerated again from the new meta-model specification. An advantage of this approach is that changes in the defini-

tion of the meta-model are easily applied to generate personalized editors, facilitating the needed flexibility for handling the inherent diversity of AmI systems. Second, regarding the modules, they are developed following a general process based on both the definition of specific templates for each target platform and procedures to retrieve information from INGENIAS models [19]. Currently, IDK incorporates modules to generate code for JADE language from design artifacts as well as documentation in HTML format for these artifacts. Finally, a module named *code uploader*, which is used to keep the code components design artifacts updated with changes made in the implementation, is also available.

3. IDK Modules for ICARO

The development of IDK modules has followed a "bottom-up" approach to support ICARO as the target platform chosen for the final implementation of a MAS application. In first place, the INGENIAS structures for specifying all concepts and their relations which are necessary to implement an application in ICARO are identified. Then, a module which automates the task of ICARO code generation from INGENIAS specifications, in line with the identified conceptual relations, is gradually developed. Finally, a new module upgrades the specification of a model when there are changes in the implementation. A detailed description of the general process for developing IDK modules can be found in [31]. The next subsections provide a description about the relations between INGENIAS and ICARO concepts, as well as the development of the modules to generate code for ICARO and to support the update of code, respectively.

3.1. Conceptual Relation between INGENIAS and ICARO

First, it is worth explaining some details of the figures that describe the relationship between INGENIAS and ICARO. The right side of the figures correspond to the notation chosen to express a fragment of a model using ICARO concepts and the left side is the notation used to express the same fragment but in terms of the INGENIAS language.

Any communication between the components implemented to develop a new executable ICARO application can be summarized as follows. First, an *event* is an entity for exchanging information between the producer of the event and potential receivers. An event is used for communication and information delivery from a resource to its agent or among agents. Thus, agents send events through their use interfaces and, in the same way, a resource also uses the use interface of an agent to send it an event. Second, an agent uses the resource use interface to request the services (methods) it offers. From our point of view, the concepts of reactive application agent and application resource used in ICARO can be modeled in INGENIAS by using the concepts of agent and application, respectively. For example, when establishing a relationship *ApplicationBelongsTo* between an agent and an application, it is understood that the agent uses the services offered by the resource (see Fig. 2). In particular, the actions that agents execute on the environment are represented by this structure. Services are modeled as application methods.

Sending information from a resource to an agent is modeled in INGENIAS by establishing a relationship *EPerceives* between the agent and the application which represents the resource (see Fig. 3). In INGENIAS, this information falls within the relationship *EPerceives* that is modeled with an event of type *ApplicationEvent* when a resource simply sends a signal to the agent. But, it is modeled with an event of type *ApplicationEventSlots* when more information has to be sent. In the latter case, the information and its type is modeled with *slots* entities.

Now, sending information among reactive agents is modeled in INGENIAS by specifying an entity of type *InteractionUnit* and relating it to the producer and consumer agents by means of the *UInitiates* and *UICollaborates* relationships, respectively (see Fig. 4). If the producer sends information, then it is included in the interaction unit through an entity of type *FrameFact* containing the necessary *slots* to transport it. Conversely, if it only needs to send a signal, then the *FrameFact* is not included in the interaction unit. Visually, it is possible to know that an interaction unit includes a *FrameFact* because it shows the "Info" attribute (the value shown is the identifier of the *FrameFact*).

Whenever an ICARO user wants to implement the behavior of a reactive agent, he/she has to create an automaton which is modeled with a state diagram. In particular, five structures available in INGENIAS "state diagram" and a criterion to name agents are needed to specify any automaton (see Fig. 5):

Figure 2. Modeling an agent using resource's services.

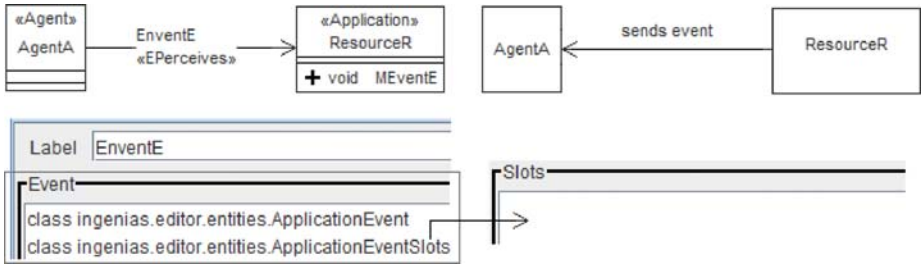

Figure 3. Modeling an agent's perception.

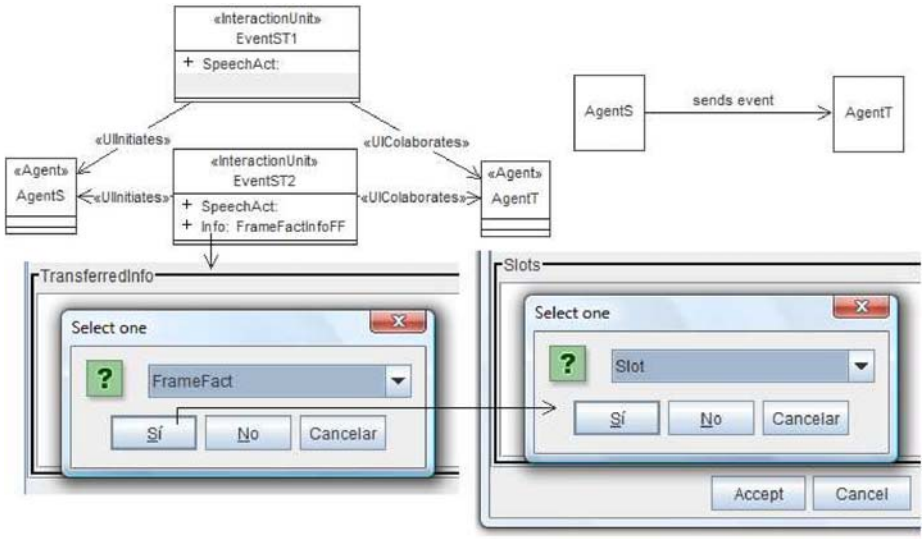

Figure 4. Modeling a communication among agents.

– A relationship is established between an "InitialNode" entity and the state to represent the initial state.
– A relationship is established between an "EndNode" entity and the state to represent a final state.
– A "WFollowGuarded" relationship is established between two different states to represent a transition; and a transition is specified using the syntax event / semantic action in its "Condition" attribute. The event represented in the state diagram is related to an *ApplicationEvent* or an *ApplicationEventSlots* entity when the event is sent by a resource (see Fig. 3). It is related to an *InteractionUnit* entity if the event is sent by an agent (see Fig. 4). The semantic action takes the same name as the task created in the components diagram.
– The IDK tool does not enable to explicitly represent relationships that cycle over the same entity, so that a fourth structure has been considered. In order to represent a transition that comes back to the same state, first a copy of the state is made, afterwards a "WFollowGuarded" relation is established from the copied state to the original state, and finally the transition is specified following the syntax described in the previous third structure.
– Universal transitions of an automaton of an ICARO reactive agent are valid for any state of an automaton. That is to say, when the event arrives, actions

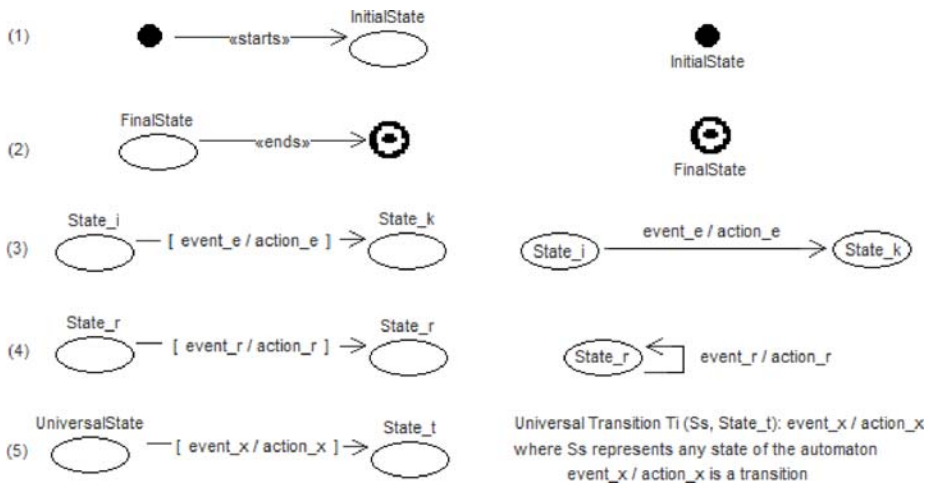

Figure 5. Modeling an agent's automaton.

are executed and the next state is reached, regardless of the automaton current state. The solution for graphically representing them in INGENIAS is to have a "UniversalState" that represent any state and takes the role of the "source" state of the universal transition. Obviously, "UniversalState" has not to be used with a different meaning.

– Finally, notice that the agent's name is assigned to the state diagram as a criterion to identify the agent's behavior.

The XML file that describes the organization of an ICARO application represents its deployment. The IDK tool offers the possibility of creating deployment diagrams. The number of instances of each type of agent is specified using entities such as *DeploymentPackage* and/or *DeploymentUnitByType*. However, the existence of an application instance is conditioned by the existence of the instance of an agent. For this reason, it is necessary to find an alternative way of independently expressing the number of instances of agents and applications, such as in ICARO. The solution is to create an environment model and to follow the following steps: (1) to copy all the agents and applications, (2) to relate them with entities of type *UMLComment*, and (3) to set the number of instances to be deployed in the attribute *Text* of *UMLComment*. Obviously, this process can be repeated over and over to create different deployment configurations.

3.2. Code Generation Module

The IDK module named (INGENIAS ICARO Framework generator) *IIF* has been developed to generate code for the ICARO framework. For this aim, the *IIFGenerator* class is extended so that its constructors possesses the templates that the *IIF* module uses in a similar way to any other IDK code generator. Moreover, the extended *IIFGenerator* class also implements the abstract methods defined in *BasicCodeGeneratorImp*. It is worth noting that the development of the *IIF* module has been simplified by defining a template for each artifact that an ICARO user has to implement (see Table 1).

The IDK templates for code generation contain source code written in the programming language of the target platform and tags to establish where the model information is used during the generation of code. The kinds of tags to be used in an IDK template is very limited [19]: *program* is the main tag of the document, *repeat* means that the text enclosed by this tag has to be copied and pasted to have a duplicate, *v* represents a variable, and *saveto* is used to save the enclosed text into a file. Therefore, it can be stated that the IDK code generation technology is more straightforward and easy to learn than other technologies for code generation, such as XSLT [39] or XPAND [21]. However, IDK exhibits a disadvantage as it does not enable developers to reuse templates. They have to copy and paste the fragments to be reused, this way hindering the code generator maintainability.

Next, the elements used by the *IIF* module to generate code for the intermediate states of an ICARO reactive agent automaton are shown. With this aim, the *automaton* template specifies the following pattern: for each (first repeat) intermediate state defined by the *intermediateState* variable, generate code for each (sec-

Table 1
Description of the templates

Template	Description
Automaton	It is used to generate an xml file with the automata agents code.
SemanticAction	It is employed to generate code for the classes that implement the agents' semantic action.
ResourceGeneratorClass	It is used to generate code for the classes that implement the resource use interfaces. The code of both methods and parameters of these classes is automatically generated as well.
ResourceUseItf	It is used to generate code for the resource use interfaces.
Deployment	It is employed to generate an xml file with the organization of the ICARO application under development.

ond repeat) transition that starts with such intermediate state.

```
@@@repeat id="intermediateStates"@@@
<state intermediateId="@@@v@@@intermediateState@@@/v@@@">
  @@@repeat id="isTransitions"@@@
   <transition input="@@@v@@@event@@@/v@@@"
               action="@@@v@@@action@@@/v@@@"
               nextState="@@@v@@@nextState@@@/v@@@"/>
  @@@/repeat@@@
</state>
@@@/repeat@@@
```

When the *IIF* module is executed using a model created with IDK, a sequence of data is generated. For instance, a sketch of the sequence of an agent automaton is shown next.

```
<repeat id="intermediateStates">
   <v id="intermediateState" entityID=""
                             attID="" >IS1</v>
   <repeat id="isTransitions">
      <v id="event" entityID="" attID="" >EV1</v>
      <v id="action" entityID="" attID="" >A1</v>
      <v id="nextState" entityID="" attID="" >NSA</v>
   </repeat>
   <repeat id="isTransitions">
      <v id="event" entityID="" attID="" >EV2</v>
      <v id="action" entityID="" attID="" >A2</v>
      <v id="nextState" entityID="" attID="" >NSB</v>
    </repeat>
</repeat>
```

Finally, it is worth noting that the *IIF* module performs a matching between the templates and the data retrieved from the model. Next, following our example, the code generated by the *IIF* module it is shown for an intermediate state and two transitions that start from it.

```
<state intermediateId="IS1">
   <transition input="EV1" action="A1"
                           nextState="NSA"/>
   <transition input="EV2" action="A2"
                           nextState="NSB"/>
</state>
```

3.3. Module to Support Code Update

Another important aspect to consider when developing a code generator is to provide developers with facilities that prevent manually written code from being overridden by subsequent generator runs. The solution in *IIF* has been to integrate a facility for marking protected regions where the developers can manually write code. The start and end of a protected region is marked by means of comments. A file can have as many protected regions as necessary, each one labeled with a unique identifier. For instance, the class implementing the agents' semantic actions has a protected region for each semantic action established in the state diagram that specifies the agent automaton. The following fragment of code shows an example of this type of region, where *ACTIONID* has to be replaced with the identifier of the related semantic action.

```
//#start_nodeIDACCION:ACTIONID <--ACTIONID--
//#end_nodeIDACCION:ACTIONID <--ACTIONID--
```

The *IIF* module uses the specification of the model to generate code. Therefore, it is necessary to store a copy of the code manually written in the protected regions. In this way, each time the *IIF* module is run, each protected region is overridden with the code manually written by the developer.

Another module in charge of synchronizing code and design, named *ICAROTCodeUploader*, has been developed. When it is executed, the design specification is updated with the regions of the code generated. This module, unlike the *IIF* module, does not require templates for its definition.

4. Case Study: Personal Health Monitoring

A case study consisting in an AmI scenario for personal health monitoring is introduced to show the effectiveness of our approach. Personal health monitoring refers to any electronic device or system that monitors a health-related aspect of a person's life on a constant basis outside of a hospital setting [25]. Some de-

tails related to our monitoring scenario are described next. There are several electronic devices used to monitor the vital signs of a patient (see Fig. 6):

- A heart rate monitor (HRM) enables to measure a person's heart rate in real time.
- A pulse oximeter (PO) enables the monitoring of the saturation of a patient's hemoglobin.
- A continuous respiratory monitor (CRM) is used for monitoring of respiratory problems.
- Ambulatory blood pressure monitoring (ABPM) is carried out to measure blood pressure at regular intervals.
- A continuous glucose monitoring system (CGM) measures glucose levels.
- A body temperature sensor measures human body temperature.
- A localization sensor allows to know where a human is.

The patient is described through the following parameters: man; 60 years old; 172 cm tall; 90 kg weight; smoker. The patient is constantly being monitored by the previously mentioned devices. Let us assume that at a given moment the monitors show the following values about his/her vital signs:

- The heart rate ranges from 80 to 90 heartbeats per minute.
- The level of oxygen saturation (SpO_2) measured by the pulse oximeter is equal to 91%, which reveals that the human presents a mild desaturation.
- The respiratory rate is 24 breaths per minute.
- The clinical systolic and diastolic blood pressures are equal to 150 and 100 mm Hg, respectively. This suggests hypertension.
- The concentration of glucose in blood is 70 mg/dl.
- The body temperature is 37.6 degrees (Celsius).

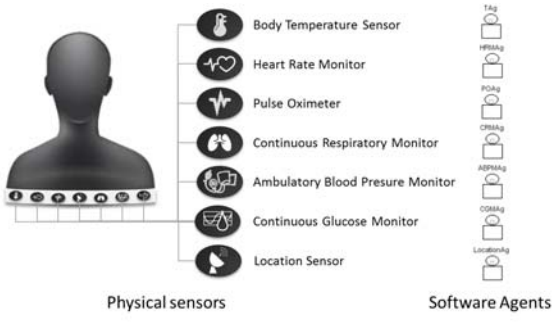

Figure 6. Health monitoring system (part of)

Although the heart rate is normal, there are some alterations in other vital signs that may infer an acute myocardial infarction (AMI). Moreover, it is known that two family members died of AMI. Therefore, a mobile intensive care unit (MICU) is immediately called to take care of him. It is supposed that there are ten MICUs available in the health care system.

The scenario describes an example of execution of an AmI personal health monitoring system. Once each type of sensor needed for monitoring patients has been identified, it can be easily modeled as a software agent [32]. For instance, Fig. 6 shows that agent "TAg" in charge of manipulating the "Body Temperature Sensor" is modeled. The scenario introduces two additional requirements, autonomy and communication, for which MAS are especially appropriate [23]. The patients live at their homes, usually suffering mobility problems, so that the AmI personal health monitoring system has to communicate with them in order to monitor their state and to autonomously reason whether an anomalous situation happens in order to act properly.

The AmI personal health monitoring system has to deal with the described scenario. For this, firstly, the entities responsible for coordination are modeled as agents, whereas entities responsible for managing persistent information are modeled as applications (see Fig. 7). Moreover, two applications are identified to denote the graphical user interfaces that interact with humans and medical professionals. Then, the communications among agents are modeled. In INGENIAS these communications are specified as interaction units that contain a *framefact* which stores the transported information in the communication (see Fig. 8). So, similar structures to those depicted in Fig. 4 are specified with IDK to represent the communications among agents.

Next, the different scenarios have to be specified for the AmI personal health monitoring system. For instance, the interaction protocol illustrated in Fig. 9 is specified to deal with the scenario described above integrating the information collected by the different devices. These agents communicate with a personal health agent which is responsible for integrating the information obtained from the devices and to evaluate if the symptoms are related to some health problems. Moreover, the personal health agent queries a user database to know additional information about the monitored human (e.g. age, gender, previous diseases and family health history). A (*UserDB*) application models the management of this database, that is, to get, delete and update user persistent information.

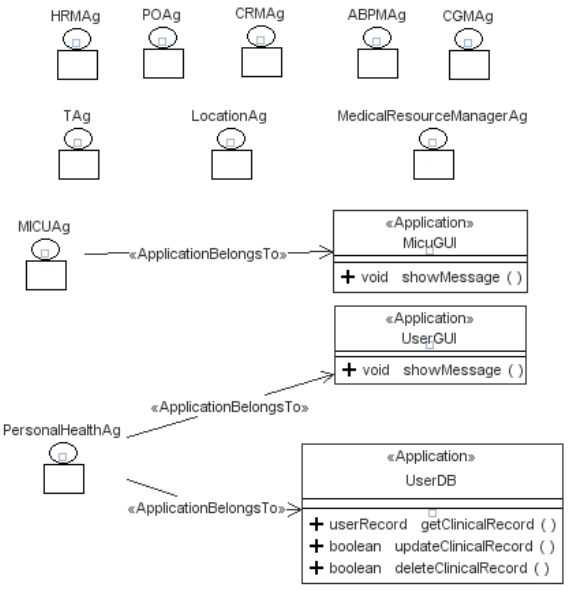

Figure 7. Environment model.

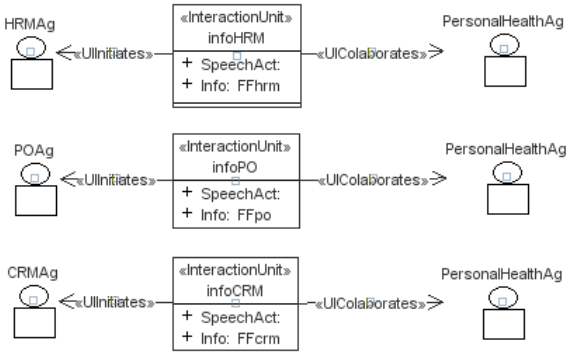

Figure 8. Partial view of the Interaction model.

The personal health agent behavior concludes that an AMI might be happening and communicates this fact to a medical resource manager agent. The last agent selects the best MICU according to the geographical location of the human.

As aforementioned, AmI systems have to monitor, reason and react. Monitoring of the health system as well its reaction can be specified through INGENIAS. Moreover, INGENIAS can be also used to model the behavior of the reactive agents through automata (state diagram). For example, the personal health agent reacts to the information related to the devices as sent by other agents. Moreover, it is responsible for reasoning if any health problem is happening through the human health state by using different rules, such as:

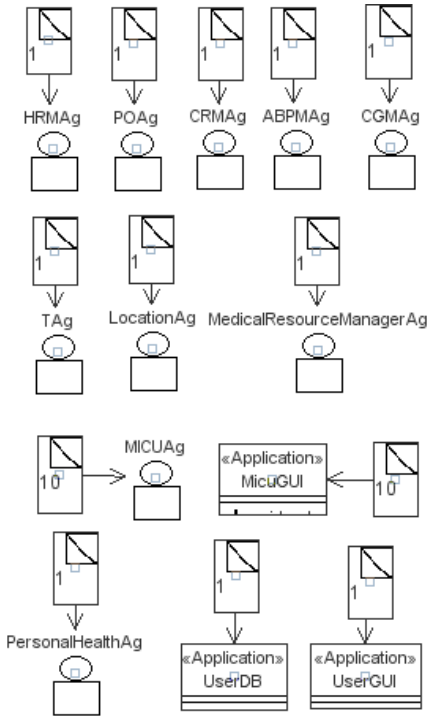

Figure 11. Deployment model.

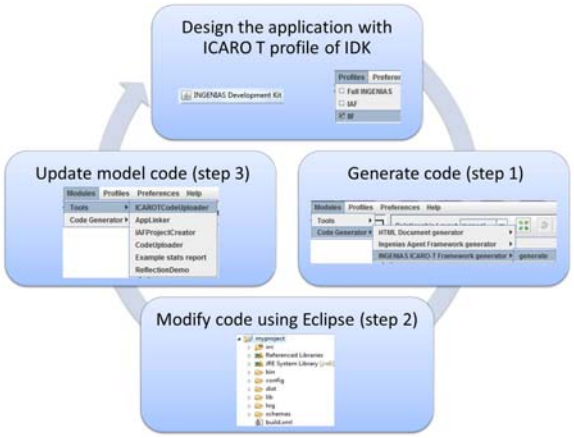

Figure 12. Development of ICARO applications using IDK.

```
If heart rate > 100 Then Tachycardia;
If heart rate < 60 Then Bradycardia;
If breaths per minute > 20 Then Tachypnea;
If breaths per minute < 12 Then Bradypnea;
...
```

So, a particular health problem is detected when an individual rule is satisfied. However, the interest of its function increases when several conditions are simultaneously satisfied, because the integration of different information sources allow to diagnose more complex

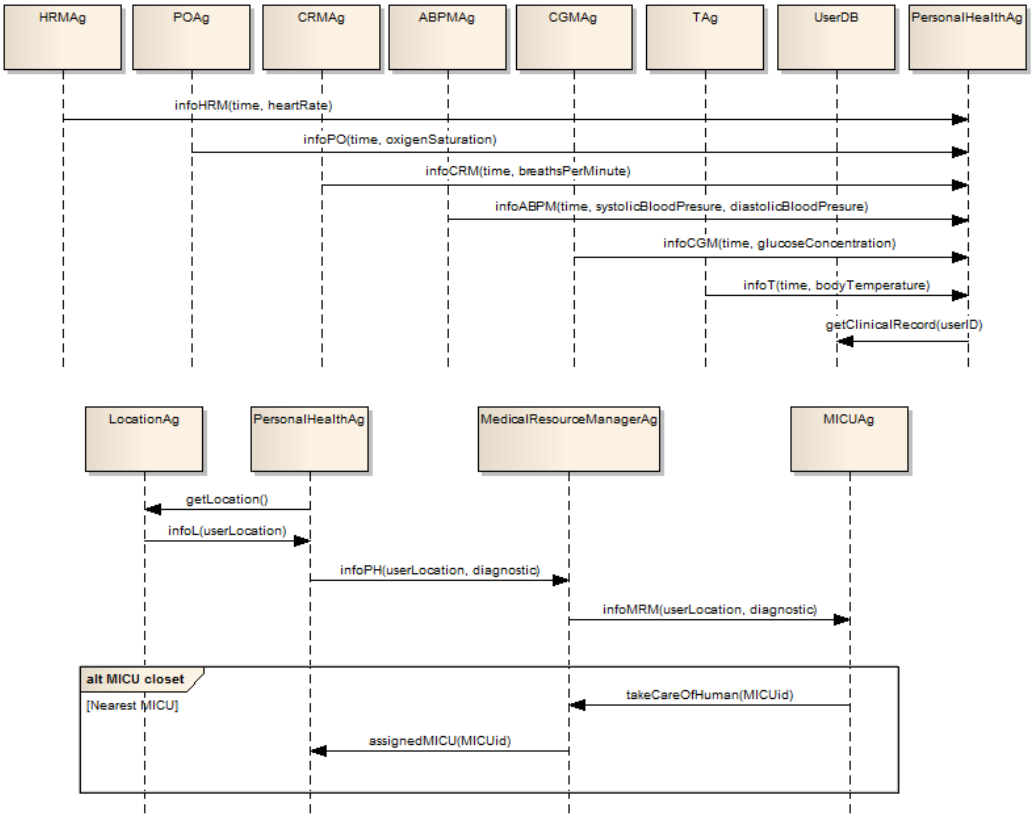

Figure 9. Interaction protocol.

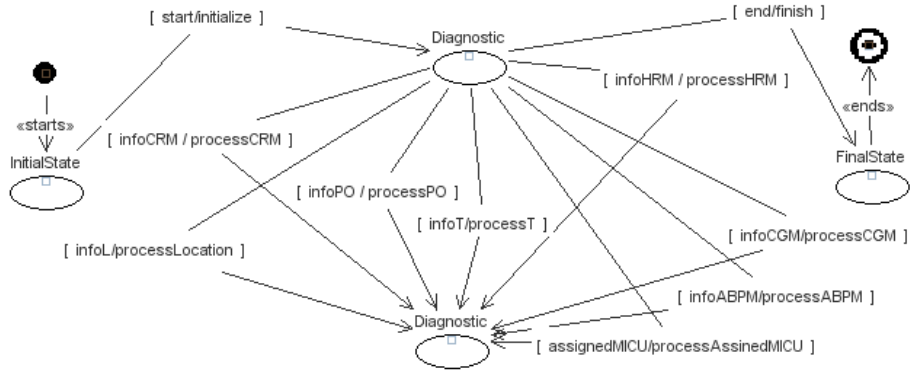

Figure 10. Automaton of personal health agent.

problems (as an AMI in the described scenario). The personal health agent holds an initial state, a final state and remains in an intermediate state to receive information sent by the agents related to the devices (see Fig. 10). Each certain time a diagnosis is issued by the agent.

It is also necessary to specify another model for the deployment of the application (see Fig. 11). For instance, this model would specify there is one instance for each type of agent related to a device, one instance of personal health agent, one instance of medical resource manager agent, ten instances of MICU agents and MicuGUI applications, and one instance for appli-

cations of type UserDB and UserGUI for the described scenario. It is worth noting that the above described models are just a partial view of the system, as they would have to be extended to consider all its requirements. For instance, this is the case in order to create new patients or to define additional reasoning.

Finally, once the different INGENIAS models are specified, the following process is carried out using the developed modules in order to obtain the final application (see Fig. 12):

1. The developer uses the INGENIAS ICARO Framework generator module (*IIF*) to automatically generate code for the design specified with the IDK tool (see Fig. 12, step 1). *IIF* generates several XML files that describe the behavior of each agent, java classes for each agent and application, and the XML file describing the application deployment.
2. After that, the developer manually inserts code into the protected regions of the generated java classes and implements the new necessary classes (see Fig. 12, step 2). Specifically, the developer only needs to develop the graphical user interfaces, knowing how an agent accesses the resources and how an agent and/or a resource sends an event to an agent to complete the implementation of the application. Let us clarify that resources code is generated from application entities specified in the detailed design model using the INGENIAS modeling language.
3. The developer uses the *ICAROTCodeUploader* module to update the model with the modifications introduced in the protected regions (see Fig. 12, step 3). It enables the developers to always keep synchronized the model and the source code, so that changes introduced in the source code are kept when the code is again regenerated from the model. Finally, the script file generated by the *IIF* module is executed by the deployment manager to launch the developed application.

5. Conclusions

This paper has shown the easiness to implement AmI applications using the IDK tools to design them as MAS. The paper has also described the two developed to generate code for the ICARO framework from the design. More specifically, the development of these two modules (code generator and code update) that provide support for the implementation of ICARO reactive agent applications has been described. These modules are integrated in the IDK tool. It is worth pointing out that the time spent learning how to develop and implement the *IIF* and the *ICAROTCodeUploader* modules described in section 3 was two months and fifteen days. This effort shows an improvement in modeling and implementing new applications in terms of productivity. The main reason is that the time necessary for coding is reduced because developers do not need to learn the structure, location and naming rules of ICARO applications files.

We would like to point out that the presented modules have been validated through their use in the development of two different applications. The first application [16] was developed to face the problem of a collection of robots patrolling around a surveillance environment. The second application [17] was developed for monitoring and controlling the normal and anomalous situations that happen when humans access to a specific area. During the development of both applications, the developer only had to write manually the body of both the resources methods and semantic actions along with some auxiliary classes. The remaining code was automatically generated by using as input the models created with the IDK.

Finally, two challenges constitute our on-going work. The first one is related to the validation of the proposal. The developed modules have shown their usefulness and suitability when used in the two aforementioned projects. They allow to reduce the time necessary to perform coding tasks and to increase the quality of the developed systems since the generated code contains no errors. Although these results are very promising, we are currently designing an experimental evaluation [12] with developers to determine the acceptance of this proposal in a close future. The second challenge is mainly related to the extension of the IDK modules to provide additional support. Currently, we are extending the IIF module to generate the event notification code.

Acknowledgements

This work is partially supported by Spanish Ministerio de Economía y Competitividad / FEDER under projects TIN2010-20845-C03-01 and TIN2012-34003 and TIN2013-47074-C2-1-R.

References

[1] Augusto J. (2007). Ambient intelligence: the confluence of pervasive computing and artificial intelligence. Intelligent Computing Everywhere, pp. 213–234. Springer.

[2] Ayala I., Amor M., Fuentes L. (2013). A model driven engineering process of platform neutral agents for ambient intelligence devices. Autonomous Agents and Multi-Agent Systems, doi:10.1007/s10458-013-9223-3.

[3] Aztiria, A., Izaguirre, A., Augusto, J.C. (2010). Learning patterns in ambient intelligence environments: a survey. Artificial Intelligence Review, vol. 34, pp. 35-51.

[4] Bellifemine F., Caire G., Greenwood D. (2007). Developing Multi-Agent Systems with JADE. John Wiley and Sons.

[5] Bordini R.H., Dastani M., Dix J., El Fallah Seghrouchni A. (2009). Multi-Agent Programming: Languages, Tools and Applications (1st ed.). Springer Publishing Company, Incorporated.

[6] Bosse T., Both F., Gerritsen C., Hoogendoorn M., Treur J. (2012). Methods for model-based reasoning within agent-based ambient intelligence applications. Knowledge-Based Systems, vol. 27, pp. 190–210.

[7] Boissier O., Bordini R., Hübner J.F., Ricci, A., Santi A. (2013). Multi-agent oriented programming with JaCaMo. Journal of Science of Computer Programming, vol. 78(6), pp. 747–761.

[8] Braubach L., Pokahr A., Lamersdorf W. (2005). Jadex: a BDI-agent system combining middleware and reasoning. Software Agent-Based Applications, Platforms and Development Kits, pp. 143–168.

[9] Chernbumroong S., Cang S., Atkinsa A., Yud H. (2013). Elderly activities recognition and classification for applications in assisted living. Expert Systems with Applications, vol. 40, pp. 1662–1674.

[10] Cook D.J., Augusto J.C., Jakkula V.R. (2009) Ambient intelligence: technologies, applications, and opportunities. Pervasive and Mobile Computing, vol. 5, pp. 277–298.

[11] Costa A., Castillo J.C., Novais P., Fernández-Caballero A., Simoes R. (2012). Sensor-driven agenda for intelligent home care of the elderly. Expert System with Applications, vol. 39, pp. 12192–12204.

[12] Davis F.D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Quarterly, 13, pp. 319–340-

[13] De Silva L. C., Morikawa C., Iskandar P.M. (2012). State of the art of smart homes. Engineering Applications of Artificial Intelligence, vol. 25, pp. 1313–1321.

[14] Ducatel K., Bogdanowicz M., Scapolo F., Leijten J., Burgelman J.C. (2010). ISTAG: scenarios for ambient intelligence in 2010. Technical Report, ISTAG.

[15] Gascueña J.M., Navarro E., Fernández-Caballero A. (2012). Model-driven engineering techniques for the development of multi-agent systems. Engineering Applications of Artificial Intelligence, vol. 25, pp. 159–173.

[16] Gascueña J.M., Navarro E., Fernández-Caballero A. (2011). VigilAgent for the development of agent-based multi-robot surveillance systems. Lecture Notes in Computer Science, vol. 6685, pp. 200–210.

[17] Gascueña J.M., Navarro E., Fernández-Caballero A. (2011). VigilAgent methodology: modeling normal and anomalous situations. Advances in Intelligent and Soft Computing, vol. 89, pp. 27–35.

[18] Gascueña J.M., Fernández-Caballero A., Garijo F.J. (2010). Using ICARO-T framework for reactive agent-based mobile robots. Advances in Intelligent and Soft Computing, vol. 70, pp. 91–101.

[19] Gómez-Sanz J.J. (2008). INGENIAS Agent Framework. Development Guide version 1.0. Technical Report, Universidad Complutense de Madrid, `http://grasia.fdi.ucm.es/main/myfiles/guida.pdf`

[20] Gómez-Sanz J.J., Fuentes R., Pavón J., García-Magariño I. (2008). INGENIAS development kit: a visual multi-agent system development environment. Proceedings of the 7th Conference on Autonomous Agents and Multi-agent Systems, pp. 1675–1676.

[21] Gronback R.C. (2009). Eclipse Modeling Project: A Domain-Specific Language Toolkit. Addison-Wesley.

[22] Lacoûture J., Gascueña J.M., Gleizes M.P., Glize P., Garijo F.J., Fernández-Caballero A. (2012). ROSACE: agent-based systems for dynamic task allocation in crisis management. Advances in Intelligent And Soft Computing, vol. 155, pp. 255–259.

[23] Leitão, P., 2009. Agent-based distributed manufacturing control: A state-of-the-art survey. Engineering Applications of Artificial Intelligence 22(7), 979–991.

[24] Mastrogiovanni F., Sgorbissa A., Zaccaria R. (2010). Activity recognition in smart homes: from specification to representation. Journal of Fuzzy and Intelligent Systems, vol. 21, pp. 33–48.

[25] Mittelstadt B., Fairweather N.B., McBride N., Shaw M. (2011). Ethical issues of personal health monitoring: a literature review. ETHICOMP 2011 Conference Proceedings, `http://www.academia.edu/attachments/31140067/download_file`

[26] Morandini M., Penserini L., Perini A. (2008). Modelling self-adaptivity: a goal-oriented approach. Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems, pp. 469–470.

[27] Morandini M., Nguyen C.D., Penserini L., Perini A., Susi A. (2011). Tropos modeling, code generation and testing with the Taom4E tool. Proceedings of the 5th International i* Workshop, pp. 172–174.

[28] Muñoz, A., Augusto, J.C., Villa, A., Botía, J.A. (2011). Design and evaluation of an ambient assisted living system based on an argumentative multi-agent system. Personal and Ubiquitous Computing, vol. 15, pp 377-387.

[29] Noel V., Arcangeli JP., Gleizes MP. (2010). Between design and implementation of multi-agent systems: A component-based two-step process. Proceedings of the 8th European Workshop on Multi-Agent Systems. `ftp://ftp.irit.fr/IRIT/SMAC/DOCUMENTS/PUBLIS/eumas-2010-noel.pdf`

[30] Padgham L., Thangarajah J., Winikoff M.: Prometheus Design Tool. Proceedings of the 23th AAAI Conference on Artificial Intelligence, pp. 1882–1883.

[31] Pavón J., Gómez-Sanz J.J., Fuentes R. (2006). Model driven development of multi-agent systems. Lecture Notes in Computer Science, vol. 4066, pp. 284–298.

[32] Pavón J., Gómez-Sanza J.J., Fernández-Caballero A., Valencia-Jiménez J.J. (2007). Development of intelligent multisensor surveillance systems with agents. Robotics and Autonomous Systems, vol. 55, pp. 892–903.

[33] Pokahr A., Braubach L. (2009). A survey of agent-oriented de-

velopment tools. Multi-Agent Programming: Languages, Tools and Applications, pp. 289–329.

[34] Ramos C., Augusto J.C., Shapiro D. (2008). Ambient intelligence - the next step for artificial intelligence. IEEE Intelligent Systems, vol. 23, pp. 15–18.

[35] Rao A.S., Georgeff M.P. (1995). BDI agents: From theory to practice. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pp. 312–319.

[36] Serrano E., Botia J. (2013). Validating ambient intelligence based ubiquitous computing systems by means of artificial societies. Information Sciences, vol. 222, pp. 3–24.

[37] Sokolova M.V., Serrano-Cuerda J., Castillo J.C., Fernández-Caballero A. (2013). A fuzzy model for human fall detection in infrared video. Journal of Fuzzy and Intelligent Systems, vol. 24, pp. 215–228.

[38] Warwas S., Hahn C. (2009). The DSML4MAS development environment. Proceedings of the 8th Conference on Autonomous Agents and Multi-agent Systems, pp. 1379–1380.

[39] Willians I. (2009). Beginning XSLT and XPaht: Transforming XML Documents and Data. Wiley Publishing Inc.

[40] Winikoff M. (2005). Jack intelligent agents: an industrial strength platform. Multi-Agent Programming Languages, Platforms Applications, pp. 175–193.

[41] Yi-bin, H., Zhang-qin, H., Jian , H. (2009). Modeling the Ambient Intelligence Application System: Concept, Software, Data, and Network. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 39, pp. 299–314.