MDD-Approach for developing Pervasive Systems based on Service-Oriented Multi-Agent Systems

Jorge Agüero, Miguel Rebollo, Carlos Carrascosa, Vicente Julián
Departamento de Sistemas Informaticos y Computacion Universitat Politecnica de Valencia
Camino de Vera S/N 46022 Valencia (Spain) {jaguero, mrebollo, carrasco, vinglada}@dsic.upv.es

KEYWORD

Multi-agent systems
Virtual organizations
Pervasive Systems
Model-Driven Development

ABSTRACT

The development of Ubiquitous or Pervasive Systems can be considered a complex task, with multiple actors, devices and different hardware environments; where it is difficult to find a compact view of all the components. This work proposes to use a MDD (Model-Driven Development) approach to facilitate the development process of Agent-Based Pervasive Systems, providing the user with a set of abstractions that ease the implementation of Pervasive Systems and the deployment of a platform for their execution. The proposal allows designing pervasive applications using high-level abstractions, avoiding the low-level implementation details and, after that, the Pervasive System deployment (with embedded agents and devices) is generated by using automatic transformations. In this way, a non-expert programmer will be able to develop Agent-Based Pervasive Systems, reducing the gap between the design and the implementation phases.

1 Introduction

The development of Ubiquitous or Pervasive Systems is a complex task, which involves multiple actors, devices and different hardware environments. So, it is difficult to find a compact view of all the components of the system and the requirements of this kind of systems are very different [ENDRES, C. 2005]. However some of them are basic: (i) integration of external devices and software systems; the services that are provided by Pervasive Systems can be supplied by physical devices and also by existing software systems, and it is essential that the system supports these issues; (ii) the isolation of the technology and the manufacturer-dependent devices, in order to facilitate the development of this kind of systems, the manufacturer-dependent devices must be well encapsulated in independent and generic functionalities.

So, it is easy to think that this paradigm implementation requires, from a designer point of view, the development of applications in different software and hardware platforms depending on the diversity of the objects in the environment. This raises big challenges. In this way, one way to implement *Ubiquitous computation* is with embedded intelligent agents. The embedded hardware (typically a computer) containing such agent is usually called embedded agent [HAGRAS, H. 2005]. Each embedded agent is an autonomous entity allowing to communicate and cooperate with other agents, as part of a Multi-Agent System (MAS).

Model-Driven Development (MDD) approach facilitates and simplifies the design process and improves the software quality for *Pervasive Systems*. It allows to re-use software and automatic transformation between models [SELIC, B. 2003]. This methodology can be applied in the development of embedded agents for ubiquitous computation, where different technologies and developing platforms coexist.

This work is centered in the deployment phase of a MDD approach for the development of embedded agents for ubiquitous computation. In this sense, a MDD approach to develop agent-based software for Pervasive Environments is

presented. This MDD approach simplifies the design and implementation of application prototypes. Our approach provides a method for the specification of *Pervasive Systems*, which allows facing the development of such systems from a higher abstraction level. The deployment over different execution platforms is achieved by means of automatic transformations among models that described entities and the environment (UML-like). The result is a simplified and homogeneous deployment process for *Agent-Based Pervasive Systems* [TAPIA, D. 2013]. Finally, our approach allows the implementation of *Pervasive Systems* over a service-based framework (OSGi-based) and it allows too agents to manage the services and the context adapting the environment. This approach gives the possibility to create more advanced and powerful *Pervasive Environments*.

This document is structured as follows. Section 2 briefly describes our MDD approach and Development process. Section 3 shows how to implement the MDD approach in order to develop *Pervasive Systems* and explains the deployment architecture. Section 4 shows a case study, in order to illustrate this approach. Finally, some conclusions are presented in section 5.

2 πVOM

One fundamental challenge when defining a meta-model is selecting which concepts or components will be included in order to model the system. A generic platform-independent meta-model of an *Agent-Based Pervasive System* is presented in this section. To do this, common elements in existing MAS and *Pervasive Systems* methodologies, have been identified and incorporated to the Computation Independent Model (CIM) level (see Figure 1). These models can be adjusted as MDD models that specify the concepts of the system, as roles, behaviors, tasks, environment, interactions or devices. The models can be used to describe an *Agent-Based Pervasive System* without focus on platform-specific details and requirements, as a Platform Independent Model (PIM). After that, it is possible to transform PIM models into Platform Specific Models (PSM). Figure 1 shows relationships between the concepts of different MDD models and their transformations.

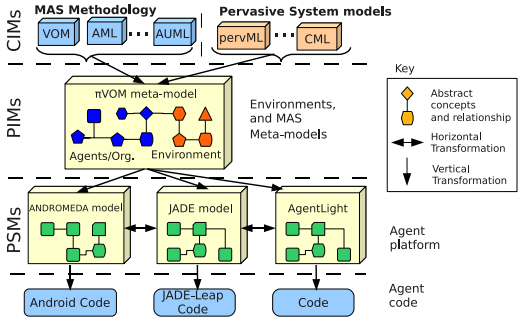

Fig. 1. MDD for Agent-Based Pervasive Systems

The proposed set of meta-models integrate different MAS modeling approaches, and it mainly focuses on the integration of Services and MAS techniques for supporting dynamical and open MAS societies [4]. This set of meta-models is called πVOM (*Platform-Independent Virtual Organization Model*). The main views of πVOM are the *Structure*, *Functionality*, *Normative*, *Agent*, and its *Environment*. Therefore, to model the characteristics of these components in our approach, five key concepts are used: *Organizational Unit*, *Service*, *Environment*, *Norm*, and *Agent* [ARGENTE, E. 2008].

According to this, πVOM is structured in five meta-models or views, which cover the above, mentioned key aspects.

These five elements describe those members (entities) that form the organization: the topology of the organization; the services and features that the organization offer; the evolution of the organization over time; the environment where the organization is situated; and the rules about the behavior of members respectively. These five elements are described in more detail in [AGUERO, J. 2010]. Next section describes the related deployment process in detail. First, a briefly description of the development process is presented.

2.1. Development process

The design process starts trying to model the agents and the environmental devices using the abstract components of the proposed meta-models (commented in the previous section). The *Pervasive System* design process is formed by a set of transformations that finally will obtain the

OSGi-Java code. In order to do these steps, a set of tools, which support the process, are required. The tools used at each stage of the design can be summarized as follows:

Step 1: At the beginning, the developer must create the different diagrams (using the EMF toolkit), which model the different devices, resources or services of the agents. To perform this step, an Eclipse IDE with a set of *plugins* is employed. These plug-ins are mainly *EMF*, *Ecore*, *GMF* and *GEF*, which allow the user to draw the models that represent the *Pervasive System*.

Step 2: Once the model has been developed, it is necessary to select in which platforms the user wants to execute the agents. This phase corresponds with the PSM model definition of each agent. To do this, it is necessary to apply a model-to-model transformation (PIM-to-PSM). This is done using the Eclipse IDE and the ATL *plug-in* incorporating the appropriated set of transformation rules. It is important to remark that the same agent model can be transformed into different specific agent platforms. Table I illustrates the agent transformations, from agent meta-model of *πVOM* to JADE-Leap [AGUERO, J. 2013]. These rules are a subset of the transformation rules needed in this phase, which are explained in detail in [AGUERO, J. 2013]. In this way, agent concepts are mapped from source models to target models, and agent components are transferred or changed from one model to another.

Rule	Concept	Transformation
1	Agent	πVOM.Agent ⇒ JADE.Agent
2	Behaviour	πVOM.Behaviour ⇒ JADE.ParallelBehaviour
3	Capability	πVOM.Capability ⇒ JADE.OneShotBehaviour
4	Task	πVOM.Task ⇒ JADE.Behaviour

Table 1. Transformation rules between agent meta-model and Jade-leap model

Step 3: After the second step, the developer must apply a transformation to convert the models into the MAS code (OSGi-based). To do this, we must use a PSM-to-code transformation. In this case, we use *MOFScript*, which is an Eclipse plug-in that uses templates to do the translation. These templates have been developed for two MAS platforms: JADE-Leap and ANDROMEDA [8]. Figure 2 illustrates how one rule is implemented using *MOFScript*. Part of the code of the rule shows the transformation of the *agent* concept.

Step 4: Finally, the process finishes by adding the necessary drivers for the different environment devices needed in the *Pervasive System*. All the functionality of physical devices are encapsulated as OSGi services, which allow agents to use them without worrying about low-level features. However, it is necessary to provide the driver/firmware/protocol of the new devices that are not in the OSGi *bundle* library. This application has a *bundle* library, which store the *EnvironmentService* (service devices) for frequent use or re-use.

```
texttransformation AGENT2ANDROMEDA (in myAgentModel:uml2)
//Rule1: Agent transformation
uml.Package::mapPackage () {
  self.ownedMember->forEach(c:uml.Class)
     if (c.name != null)
       if (c.name = Agent) c.outputGeneralization()   }
uml.Class::outputGeneralization(){
   file (package_dir + self.name + ext)
   self.classPackage()
   self.standardClassImport ()
   self.standardClassHeaderComment ()
   <% public class %> self.name <% extends Agent { %>
      self.classConstructor()
      <% // Attributes  %>
      self.ownedAttribute->forEach(p : uml.Property) {
            p.classPrivateAttribute()
      } newline(2) ...
```

Fig. 2. *Agent* translation using MOFScript

3 Deployment Process

As before commented, this work proposes to use a homogeneous and unified model for implementing *Agent-Based Pervasive Systems* permitting its translation into different execution MAS platforms through MDD, in which agents act/perceive about the environment and thereby manage/control the *Pervasive System*. This means that the user can design a *Pervasive Systems* with a unified, intuitive, visual model, i.e., with a high level of abstraction. Then, the user can get the agent code automatically using MDD with minimal user intervention. Finally, the drivers or firmware must be added to support environmental devices. These drivers will be encapsulated within a service (an OSGi service), to export their functionality as a high-level abstraction (which will be managed by agents). Finally, this code should be compiled for execution over an OSGi framework, as it is shown in Figure 3.

In order to do this final step, it is necessary to employ an appropriated deployment platform, which includes all the computational resources needed to give support to systems designed according to the proposed development process.

Implementing *Pervasive Systems* is a challenging and exciting task, since a solid background knowledge about how to implement this kind of systems do not exist. Many research efforts are currently being developed on prototype implementations [ENDRES, C. 2005]. However, some *Pervasive System* prototypes share a common architectural style, which correctly fits the requirements of these systems.

3.1. Deployment Platform

As above commented, a deployment platform is presented in this section. The requirements of *Pervasive Systems* are basically two: (i) is essential that the *Pervasive System* must support the integration of services provided by external devices and software systems (as other services); (ii) the isolation of the low-level abstraction of the devices (manufacturer dependent), the environmental devices must be well encapsulated in independent and generic functionalities. Therefore, an architecture style that meets these requirements is the well-known layered architecture [ENDRES, C. 2005]. By means of this architecture, the system elements are organized in different levels with well-defined responsibilities. Our proposal follows this architecture style. Figure 4 shows our deployment platform, which is a framework based on OSGi technology. The main layers of this deployment platform are:

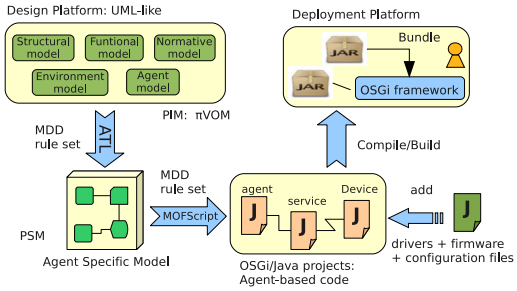

Fig. 3. Implementing Agent-based Pervasive Systems using MDD transformations

Physical layer, has the resources/devices that are perceived in the environment. Fully represents the real world, where the *Pervasive System* is located.

Sensor layer, has the responsibility of accessing to physical devices, through actuators and sensors, which allow to change or read the state of the devices.

Environment layer, has the responsibility of encapsulating the manufacturer dependent technology of the environment devices. The drivers that conform this level directly export their functionalities through a *bundle*. The *bundles*, which manage similar devices of software systems from different technology or vendors, are implemented as a common interface, in order to provide a uniform way of communicating within the environment devices.

Service layer, provides the system functionality, offering services that the *Pervasive System* must supply. The services are provided by the devices located in the physical world, and by the MAS entities (agent or organizational unit). Also, at this level the services can be *single* or *composite services*, which are formed by the composition of other services.

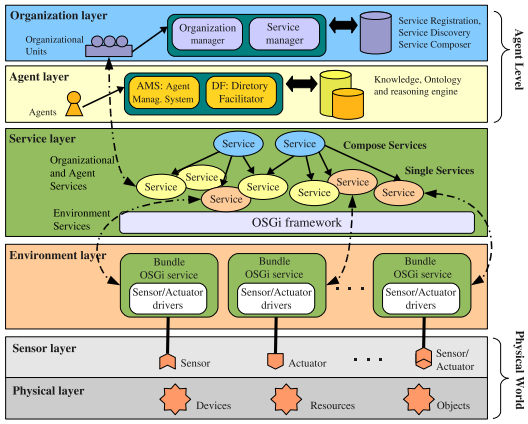

Fig. 4. Pervasive System Architecture Proposal

Agent layer, supports the mechanisms in order to register, de-register and discovery of agents. In this layer the agents work together through different interactions to support complex tasks in a collaborative and dynamic way. This layer also supports the information management (*knowledge*) and the needed *knowledge models*, including the *reasoning engine* and *ontologies* needed by the agents. Furthermore, this layer provides the necessary mechanisms to support the communications and needed languages used by agents, such as FIPA-ACL.

Organizational layer, is used as a regulatory framework for the coordination, communication, and interaction among different computational entities. This layer is formed by a set of individuals and institutions that need to coordinate resources and services across institutional boundaries. This layer supports high-level interoperability to integrate diverse

information systems in order to share knowledge and facilitate collaboration among entities. This layer is an open system formed by the grouping and collaboration of heterogeneous entities. From a technical view, these functionalities are obtained using the THOMAS platform [CARRASCOSA, C. 2009], which consists basically of a set of modular services that enable the development of agent-based organizations in open environments.

4 Implementing an Application Example

In order to illustrate this approach, a case study for an **Intelligent Mall** is employed. The intelligent Mall allows mobile users to know the product offerings for a Mall based on the user profile (preferences), ie, the user receives recommendations for possible purchase and use of Mall's Services. The Mall example is an application that facilitates the interconnection between clients (individuals, consumers) and providers (shops, Cinemas, Fast food shops); delimiting services that each one can request or offer. The system controls, which service each agent must provide. Internal functionality of these services is responsibility of provider agents. However, the system imposes some restrictions about service profiles, service requesting orders and service results.

The proposed system provides wireless data services, which allows mobile devices (mobile phones, netbooks, personal digital assistants-PDAs-) to communicate with each other and with a number of servers. The network architecture provides access to wireless Services for users equipped with mobile wireless devices, via a set of access points deployed in key points around of the Mall (usually shops which offer services and products). This architecture is built upon a number of wireless communication standards -WLAN (WiFi), WPAN (Bluetooth), …- which are employed to deliver these services to registered users. This network architecture is shown in Figure 5.

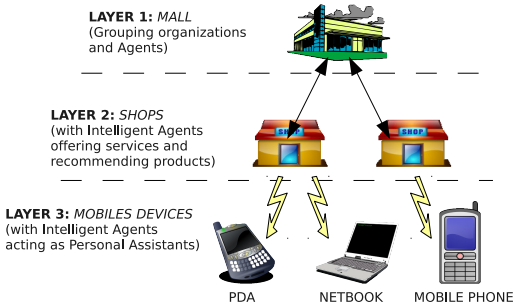

Fig. 5. The Mall network architecture

The *third layer* is composed of mobile devices equipped with intelligent agents that act as Personal Assistants (PA) for the users. The *second layer* consists of the shop agents, which act as access points or gateways for the supply of services and products. These shop agents are located in various points of the Mall and facilitate access to mobile users. Due architecture systems the network access can be interrupted at any time. The *first layer* is composed for the Mall, which acts as a "container" of the different shops as a complex organization. Its real function is to allow the creation and registration of shops and their services, and, in addition to control and synchronize the whole information system.

4.1. Intelligent Mall through a pervasive system

The physical environment is composed of a vast and rich infrastructure of hidden electronic devices and components, which may differ significantly. This approach uses different technologies adopting open technologies (services and standards, by example, standard communication protocols) reducing the seriousness of any interoperability issues that may arise at some future point. Therefore, the Mall environment will support a number of services, and one function that it could fulfill concerns the composition of elementary services into sophisticated services that would ultimately enhance the quality of the clients' experience. Figure 6, shows how the Mall components (and technologies) are integrated and how they interact with the environment.

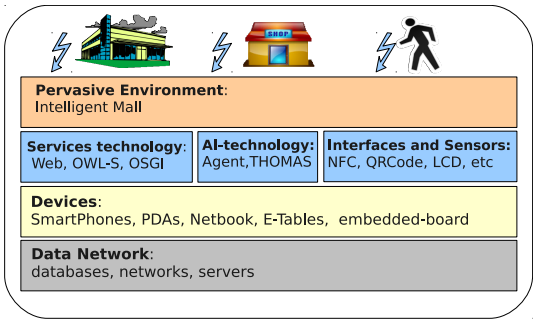

Fig. 6. The Mall modeled as a Pervasive system

4.2. Deployment Platform for the Mall

Some agent platforms (and others technologies) are needed to implement the Mall example. The Mall is designed as a virtual organization supporting an open agent society. As agents can enter and leave the organization at any time, the THOMAS platform is used. Agents using the *services and products* of the Mall have been designed as embedded agents on mobile phone by means of ANDROMEDA and JADE-Leap. These technologies are described below briefly. Figure 7 shows as the Intelligent Mall actors (and technologies) are coupled on the deployment platform.

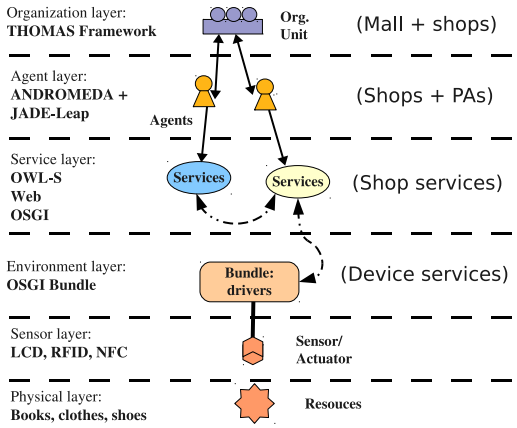

Fig. 7. Intelligent Mall over deployment platform

1. Organizational Layer: THOMAS framework.
THOMAS [CARRASCOSA, C. 2009] (MeTHods, Techniques and Tools for Open Multi-Agent Systems), is an open Multi-Agent System architecture consisting of a related set of modules that are suitable for the development of systems applied in environments that working as a "society". The use of THOMAS will give us the necessary tools for the deployment of shop and personal assistants agents into a virtual organization infrastructure. This infrastructure allows agents to dynamically enter or exit in the organization, to assign roles to each agent and to include rules or norms that agents must fulfill.

2. Agent Layer: Embedded Agent Platforms.
With respect to the agent deployment, we will use two agent platforms:

- **ANDROMEDA** [1] (ANDROid eMbeddED Agent platform) [AGUERO, J. 2008] is an agent platform specifically oriented to embedded agents over the *Android*[2] operating system. *Android* can be seen as a software system specifically designed for mobile devices, which includes an operating system, a middleware and key applications. ANDROMEDA platform includes all the abstract concepts of the πVOM agent meta-model. The implementation was done using the main API components of *Android* (SDK 1.6). The agents run as any application of *Android*, because ANDROMEDA platform can be interpreted as a new layer that is inserted into the *Android* architecture, as shown in [9].
- **JADE**[3] [BELLIFEMINE, F, 1999] one of the most popular or relevant platforms that support the agents execution, widely used because it provides programming concepts that simplify the MAS implementation. JADE is FIPA compliant in the communication infrastructure between agents. Moreover, **JADE-Leap** (JADE Light Extensible Agent Platform) is a JADE version used to implement embedded agents, which running over J2ME (Java Micro Edition).

3. Service Layer. This layer can provide various types of services to facilitate and improve the client experience on the Mall. The types of services offered by the Mall can be grouped into three categories (depending on the type of service, the role of client and provider may change): (i) Recommendation Service; (ii) Instant Information Service; (iii) Adaptive Service.

[1] http://www.gti-ia.upv.es/sma/tools/Andromeda/
[2] Android System, http://code.google.com/android/
[3] http://jade.tilab.com/

The implementation of these services is based on the use of OWL-S technology, which enables clients (PAs) and Shops (Shop agents) to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints.
For example, Figure 8 shows the protocol service of a clothing store. The customer comes to the store, then the system registers it (if not already registered), and then the client receives the services from store. The client requests the daily specials, and the store responds with special offers products that fit the customer's style.

4. Environment Layer. This layer has the responsibility of encapsulating the devices drivers (dependent manufacturer) in a single and generic service. This encapsulation is performed by OSGi technology (using a *Bundle*). The *Bundle* implemented a common interface, in order to provide a uniform way of communicating within the devices driver. The *Bundles* runs over embedded card (the BeagleBoard [4] in this approach). The BeagleBoard is a low-power open-source hardware singleboard computer. The BeagleBoard is an embedded computer boards based on TI's ARM.

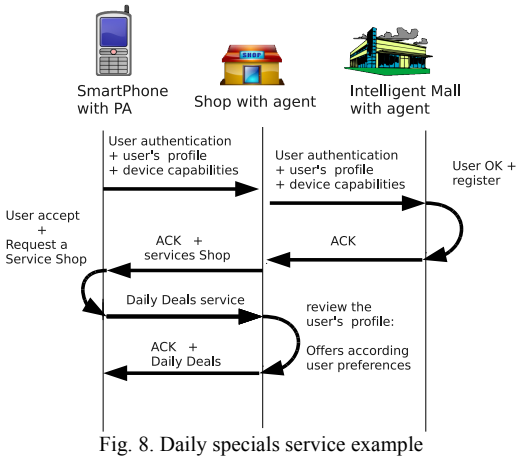

Fig. 8. Daily specials service example

5. Sensor and Physical Layers. These layers contain the sensors and actuators of used on the Mall, such as: LCD screens, RFID cards, NFC cards, QR codes, automatic locks, parking barriers, etc. Each shop has one or more embedded card that can run an embedded agent (with ANDROMEDA over BeagleBoard), whose responsibility is the sensors management.

4.3. Implementing the embedded agents

After describing the technology employed, this section presents the design and implementation of the case study. Structural definition of the system corresponds to identification of agents, roles, organizational units, norms, etc.. This case study is modeled as an organization (Mall) inside which there are different organizational units (shops) that represent group of agents. Each unit is dedicated to offer services and recommend products.

The Mall system is organized in such a way that if shop cannot fully satisfy the user service request, the request is forwarded to the other shops Center. Each Service is delivered in the most appropriate, quickest and cheapest way to each user according to his current individual location and mobile devices capabilities (specified in the user profile). When a mobile user enters within the range of a shop (access point), the intelligent agent (Personal Assistants) installed in the mobile device and the shop mutually discovers each other. The Personal Assistants sends the user's information to the shop for user register and authentication. This information includes a description of the mobile device currently being used and the user profile.
Now, when the user is successfully authenticated and authorized. The user profile is analyzed by the Shop for current user preferences and device capabilities. Then the Shops compiles a list of applicable services and products from its Service/Products Catalog and sends this to the Personal Assistants for its knowledge. The Personal Assistants shows the information regarding these services and products to the user who makes a choice and selects (makes a request for) the service/products he wishes to use. The PA forwards the user service request to the Shop, which instantiates the service.

The agent implementation starts by defining the different components used in the agent, these components are part of *agent-π* meta-model. In this case, we only describe agents running on mobile phones using ANDROMEDA or JADE-Leap platforms, due to space limitations of article. These agents (clients and consumers)

[4] http://beagleboard.org

receive service information of the Mall's shops (which depends of the user profile sent by the Personal Assistant). The Shops responds with different options for products and services to the *client* agent, and then the user must select his best option. This interaction and dialog process is shown in Figure 9.

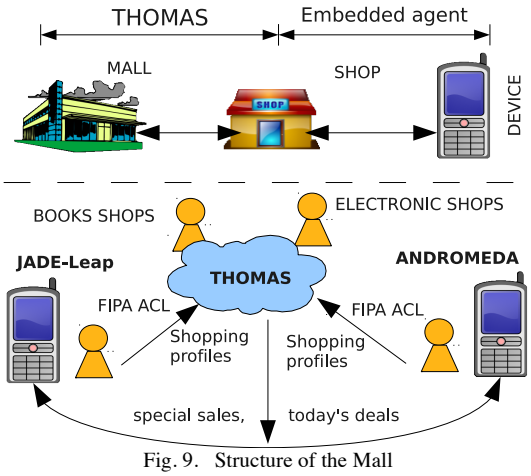

Fig. 9. Structure of the Mall

The *client* agent model represents a consumer in the system. This agent has various behaviors that allow managing services offered by the shops. For reason of brevity the paper only explains the two most important behaviors of the client agent. These two most important behaviors allow the agent to acquire the role of client and to discover the shops and send profiles. In Figure 10 these two behaviors can be observed. *Behaviour* `ShopDiscover` allows the agent to perform the activities necessary to find a shop and send the user profiles, while the other *Behaviour* (`ShopClient`) does the necessary tasks to use service and view the products offered.

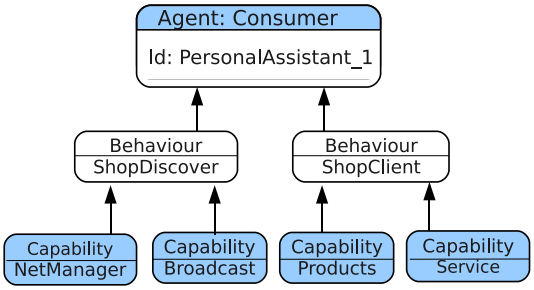

Fig. 10. Embedded agent model

These behaviors are composed of two individual capabilities. As an example the *Behaviour* `ShopDiscover` includes the *Capabilities* `NetManager` and `Broadcast`. These *Capabilities* realize the actions of looking for a shop and send the user profiles respectively. The *Behaviour* `ShopClient` can be designed in similar way, as can be seen in Figure 10.

Once we know the different components of the system, the design continues with the creation of embedded agents and the Mall system, which is modeled as an organization that will be executed over the THOMAS platform (how is modeled this organization is out of the scope of this paper, for more information please refer to [CARRASCOSA, C. 2009]). This step is accomplished using the EMF Toolkit that allows users to "draw"/design (UML-Like) all the agent components according to the *agent*-π meta-model. Once the agent model is designed, EMF Toolkit allows verifying the designed system (in a semantic and syntactic way). Figure 11 shows a snapshot of how to design the components of the client agent model and his correspondence with the *agent*-π meta-model.

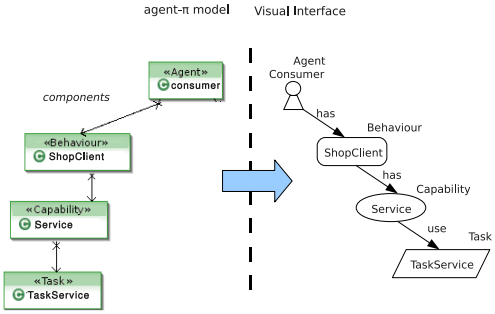

Fig. 11. Designing the client agent

After this, the next step is to generate agent code using this toolkit. The designer must decide in which mobile platform will be executed each agent. In this case, the *client* agent is selected to be executed over an agent platform: ANDROMEDA or JADE-Leap. **ANDROMEDA** platform was designed using the same concepts of the *agent* model. So, an agent in ANDROMEDA is implemented using the same concepts employed in the abstract model. This design greatly simplifies the automatic transformation between PIM and PSM, because the ANDROMEDA PSM is very similar to the πVOM agent model. Thus, the second step of the MDD process is not necessary. In this case, the needed transformation rules are mainly model-to-text, generating directly ANDROMEDA code which can be combined with additional user code. The code template

generated by the transformations is shown in Figure 12.

```
public class my_Consumer extends Agent {
     public void init(){
        . . .
      //define agent components
                                          Behaviour
      Behaviour my_ShopClient= new Behaviour("ShopClient");
                                          Capability
      Capability my_Service =  new Capability("Service");

       Task my_TaskService =new Task("TaskService");

        . . .
        //add objects into of the agent components
        my_Service.addTaskRun(my_TaskService);
        my_ShopClient.add(my_Service);
        addbehav(my_ShopClient);
    }  // ----------- end init()
                                                Task
    class my_TaskService extends Task{
       doIt {
         //here the user write the  code
       }  // ----------- end doIt
       . . .
    }  // ----------- end Class Task

    . . .
}  // ----------- end Class Agent
```

Fig. 12. Code template of ANDROMEDA Agent

The transformation from agent models to **JADE-Leap** code must be done employing the two phases previously commented: (i) the first phase translates from the PIM model to JADE-Leap PSM mode, and allows to obtain a correspondence among the abstract concepts of the agent-π model to JADE-Leap concepts; (ii) the second phase allows to translate the JADE-Leap PSM models obtained in the previous phase into executable JADE-Leap code.

```
public class my_Consumer extends Agent {
   ...
   protected void setup(){                          Behaviour
      ParallelBehaviour my_ShopClient =
         new ParallelBehaviour( ParallelBehaviour.WHEN_ALL );

      my_ShopClient.addSubBehaviour( new my_Service(this) );
      ...
      addBehaviour(my_ShopClient);
    }  // ----------- end setup()
                                                Capability
     class my_Service extends OneShotBehaviour {
         public my_Service(Agent a) {
           super(a); }

         public void action() {
           ... //check condition == true
           addBehaviour(new my_TaskService(this) );
         }
     } // ----------- end OneShotBehaviour
     ...                                            Task
     class my_TaskService extends Behaviour {
         public my_TaskService(Agent a) {
            super(a); }

         public void action()  {
             //...this is where the code Task goes !!
         }
     } // ----------- end Behaviour
} // ----------- end class Agent
```

Fig. 13. Code template of JADE-Leap Agent

In summary, this transformation process converts the `Behaviour` of agent-π in `ParallelBehaviour` of JADE-Leap. But also it translates from `Capability` to `OneShotBehaviour` and from `Task` to `Behaviour`. The code template generated by this process is shown in Figure 13.

Finally, the code, generated automatically by the tool, can be edited by the user to complete the *client* agent development with the specific details of the execution platform. Figure 14 shows the *client* agent running over the ANDROMEDA and JADE-Leap platform (executed over the mobile phone emulator). The Personal Assistant receives discounts in electronics products (according to the preferences of the user profile).

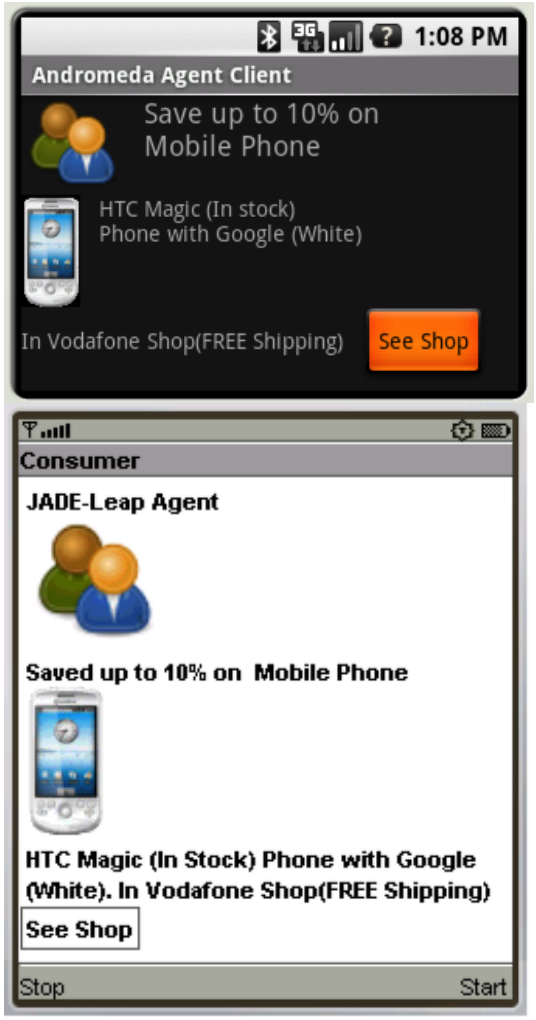

Fig. 14. Agents on mobile phones

5 Conclusions

This work presents the application of the ideas proposed by the MDD for the design of Agent-Based Pervasive Systems. Although the use of

MDD refers primarily to methodologies of object-oriented software, it was verified that the approach could be adopted in the development of Agent-Based Pervasive Systems. Concretely, a layered deployment architecture which is part of a Model Driven Development specifically designed for this kind of systems has been presented in this work. This development process allows designing pervasive applications using high-level abstractions, avoiding the low-level implementation details and, after that, the Pervasive System deployment (with embedded agents and devices) is generated by using automatic transformations. Moreover, a case study has been presented in order to illustrate the advantages of this proposal.

6 Acknowledgment

This work is supported by the MINECO /FEDER grant TIN2012-36586-C03-01 of the Spanish government.

7 References

[ENDRES, C. 2005] C. Endres, A. Butz, and A. MacWilliams, "A survey of software infrastructures and frameworks for ubiquitous computing," Mobile Information Systems, vol. 1, no. 1, pp. 41–80, 2005.

[HAGRAS, H. 2005] H. Hagras, V. Callaghan, and M. Colley, "Intelligent Embedded Agents," Information Sciences, vol. 171, no. 4, pp. 289 – 292, 05 2005.

[TAPIA, D. 2013] Dante I. Tapia, Juan A. Fraile, Sara Rodríguez, Ricardo S. Alonso, Juan M. Corchado, Integrating hardware agents into an enhanced multi-agent architecture for Ambient Intelligence systems, Information Sciences, Volume 222, 2013, Pages 47-65

[SELIC, B. 2003] B. Selic, "The pragmatics of model-driven development," IEEE software, vol. 20, no. 5, pp. 19–25, 2003.

[ARGENTE, E. 2008] E. Argente, V. Julian, and V. Botti, "MAS Modelling based on Organizations," in 9th Int. Workshop on Agent Oriented Software Engineering (AOSE08), 2008, pp. 1–12.

[AGUERO, J. 2010] J. Agüero, M. Rebollo, C. Carrascosa, and V. Julian, "MDD for Virtual Organization design," in Trends in International conference on Practical Applications of agents and multiagent systems (PAAMS2010), Springer-Verlag, Ed., vol. 71, 2010, pp. 9–17.

[AGUERO, J. 2013] J. Agüero, M. Rebollo, C. Carrascosa, and V. Julian, "Towards the development of agent-based organizations through MDD," International Journal on Artificial Intelligence Tools (IJAIT), DOI: 10.1142/S0218213013500024, pp. 1–34, 2013.

[CARRASCOSA, C. 2009] C. Carrascosa, A. Giret, V. Julian, M. Rebollo, E. Argente, and V. Botti, "Service Oriented Multi-agent Systems: An open architecture," in Autonomous Agents and Multiagent Systems (AAMAS), 2009, pp. 1–2.

[ARGENTE, E. 2011] E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julian, and M. Rebollo, "An abstract architecture for virtual organizations: The THOMAS approach," Knowledge and Information Systems, vol. 29, no. 2, pp. 379–403, 2011.

[AGUERO, J. 2008] J. Agüero, M. Rebollo, C. Carrascosa, and V. Julian, "Does Android Dream with Intelligent Agents?" in International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008), vol. 50, 2008, pp. 194–204.

[BELLIFEMINE, F, 1999] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE - A FIPA-compliant agent framework," in Proceedings of the Practical Applications of Intelligent Agents, 1999. [Online]. Available: http://jmvidal.cse.sc.edu/library/jade.pdf