A generic agent-oriented design approach applied to a collaborative system

Sara Maalal¹, Wafaa Dachry² and Malika Addou³

¹Hassan II University, Higher National School of Electricity and Mechanics ENSEM,
Architecture System Team – LISER, Eljadida Road, BP 8118 Oasis, Casablanca, Morocco

²Hassan II University, Faculty of Sciences,
Department of Mathematics and Computer, Casablanca, Morocco

³Hassan II University, Hassania School of Public Works EHTP,
Architecture System Team – LISER, Eljadida Road, BP 8108 Oasis, Casablanca, Morocco

Abstract: Agent software paradigm allows implementing large and complex distributed applications. In order to assist analyzing, conception and development or implementation phases of multi-agent systems, we’ve tried to present a practical application of a generic and scalable method of MAS with a component-oriented architecture and agent-based approach that allows MDA to generate source code from a given model. We have designed a generic class diagram as a class meta-model of different agents of a MAS [1]. Then we have applied it on a collaborative platform based on MAS supply chain which ensures cooperation and information logging and sharing between partners in real time. We have finally generated the model source code developed using an open source tool called AndroMDA. This agent-based and evolutionary approach enhances developments modularity and genericity and promotes their reusability in future developments [2].

Keywords: Multi-agents Systems (MAS), Logistic Systems, Model Driven Architecture (MDA), Agent Unified Modelling Language (AUML), Collaborative information system.

1. INTRODUCTION

Currently the computer systems are increasingly complex, often distributed over several sites and consist of software interacting with each other or with humans. The need for model human behavior in specific computer programs has prompted officials to use technology that affected the last decade and whose movements are very remarkable. In this context, designing multi-agent systems (MAS) is complex because they require the inclusion of several parts of the system which can often be approached from different angles. We must identify and analyze all system problems to find models for multi-agents to implement and integrate them into a coherent system. This is the software engineering and well justifies the use of a method of analysis, design and development of multi-agents systems [2].

This paper describes a practical example of a new generic model designed for modeling multi-agent systems and based on a class diagram, defining the different types of agents [3] and meeting our needs for development and testing of MAS applications.

Section 2 defines agents, multi-agent systems and agent types. Section 3 summaries the prior work done on designing multi-agent systems. Section 4 defines the MDA approach and presents the architecture of the project generated by AndroMDA. In Section 5, we describe the components of our AUML generic class diagram. This diagram will be then translated to an UML diagram in order to be used by the AndroMDA tool. These diagrams will primarily serve to analyze and design the related information system with more precision by using the agent technology, and secondly, they will be used to generate some code sources for modification by developers. Section 6 presents the application example of our proposed approach. We provide each description of agents composing the collaborative system example, and its resulted AUML and UML diagrams. We conclude the research in Section 7.

2. MULTI-AGENT SYSTEMS

2.1 Definitions

- An agent is a computer system within an environment and with an autonomous behavior made for achieving the objectives that were set during its design [4].
- A multi-agents system is a system that contains a set of agents that interact with communications protocols and are able to act on their environment. Different agents have different spheres of influence, in the sense that they have control (or at least can influence) on different parts of the environment. These spheres of influence may overlap in some cases; the fact that they coincide may cause dependencies reports between agents [5].

The MAS can be used in several application areas such as e-commerce, economic systems, distributed information systems, organizations...

2.2 Types of agent

Starting from the definitions cited above, we can identify the following agent types [3]:
The reactive agent is often described as not being "clever" by itself. It is a very simple component that perceives the environment and is able to act on it. Its capacity meets mode only stimulus-action that can be considered a form of communication.

The cognitive agent is an agent more or less intelligent, mainly characterized by a symbolic representation of knowledge and mental concepts. It has a partial representation of the environment, explicit goals, it is capable of planning their behavior, remember his past actions, communicate by sending messages, negotiate, etc.

The intentional agent or BDI (Belief, Desire and Intention) is an intelligent agent that applies the model of human intelligence and human perspective on the world using mental concepts such as knowledge, beliefs, intentions, desires, choices, commitments. Its behavior can be provided by the award of beliefs, desires and intentions.

The rational agent is an agent that acts in a manner allowing it to get the most success in achieving the tasks that were assigned to him. To this end, we must have measure of performance, if possible objective associated with a particular task that the agent should run.

The adaptive agent is an agent that adapts to any changes that the environment can have. He is very intelligent as he is able to change its objectives and its knowledge base when they change.

The communicative agent is an agent that is used to communicate information to all around him. This information can be made of his own perceptions as it may be transmitted by other agents.

We’ll name “method” the approach to use a rigorous process for generating a set of models that describe various aspects of software being developed using a well-defined notation.

To this end, several software engineering paradigms have been proposed, such as object-oriented design patterns, various software architectures. These paradigms fail especially when it concerns the development of complex distributed systems for two reasons: the interactions between the various entities are defined in a too rigid way and there is no mechanism complex enough to represent the organizational structure system [6, 3]. The paradigm of agents and multi-agent systems can be a good answer to these problems, because the agent-oriented approaches significantly increase our ability to model, design and build complex distributed systems [7, 3].

There are many methodologies for analysis and design of multi-agent systems. We cite below some examples of existing methodologies [8]:

- The AAII methodology was developed based on the experience accumulated during the construction of BDI systems. In this methodology, we have a set of templates that, when they have been fully elaborated, define the specifications of agents such as desires, beliefs and intentions [9].

- The first version of Gaia methodology, which modeled agents from the object-oriented point of view, was revisited 3 years later by the same authors in order to represent a multi-agent systems MAS as an organized society of individuals [10]. In fact, the agent entity, which is a central element of the meta-model of Gaia, can play one or more roles. A role is a specific behavior to be played by an agent (or kind of agents), defined in term of permissions, responsibilities, activities, and interactions with other roles. When playing a role, an agent updates its behavior in terms of services that can be activated according to some specific pre- and post- conditions. In addition, a role is decomposed in several protocols when agents need to communicate some data. The environment abstraction specifies all the entities and resources a multi-agent system may interact with, restricting the interactions by means of the permitted actions [1].

The Gaia methodology gives the possibility to design MAS using an organizational paradigm and to traverse systematically the path that begins by setting

3. MOST KNOWN DESIGN METHODOLOGIES – STATE OF THE ART

Building high quality software for real-world applications is a difficult task because of the large number and the flexibility of components but also because of the complexity of interconnections required. The role of software engineering is precisely that of providing methodologies that can facilitate control of this complexity. A methodology by definition can facilitate the process of engineering systems. It consists of guides that cover the entire lifecycle of software development. Some are technical guides; others are managing the project [5, 3].

Figure 1 Types of agents
out the demands of the problem and to lead to a fairly detailed and immediate implementation [9]. Gaia permits to design a hierarchical non-overlapping structure of agents with a limited depth. From the organizational point of view, agents form teams as they belong to a unique organization, they can explicitly communicate with other agents within the same organization by means of collaborations, and organizations can communicate between them by means of interactions. If inter-organization communication is omitted, coalitions and congregations may also be modeled [1]. However, this methodology is somewhat limited since we can describe MAS with different architectures of agents [9].

• The main contribution of MESSAGE was the definition of meta-models for specification of the elements that can be used to describe each of the aspects that constitute a multi-agent system (MAS) from five viewpoints: organization, agents, goals/tasks, interactions and domain. MESSAGE adopted the Unified Process and centered on analysis and design phases of development [11].

• INGENIAS starts from the results of MESSAGE and provides a notation to guide the development process of a MAS from analysis to implementation [12, 13]. It is both a methodology and a set of tools for development of multi-agent systems (MAS). As a methodology, it tries to integrate results from other proposals and considers the MAS from five complementary viewpoints: organization, agent, tasks/goals, interactions, and environment. It is supported by a set of tools for modeling (graphical editor), documentation and code generation (for different agent platforms). The INGENIAS methodology does not explicitly model social norms, although they are implicit in the organizational viewpoint. Organizational dynamics are not considered i.e., how agents can join or leave the system, how they can form groups dynamically, what their life-cycle is, etc [14]. The authors have developed an agent-oriented software tool called INGENIAS Development Kit (IDK) [15]. It allows to edit consistent models (according to INGENIAS specification) and to generate documented code in different languages such as JADE [16], Robocode, Servlets or Gracias Agents [1].

• Multi-agent systems Software Engineering (MaSE) is a start-to-end methodology that covers from the analysis to the implementation of a MAS [17]. The main goal of MaSE is to guide a designer through the software life-cycle from a documented specification to an implemented agent system, with no dependency of a particular MAS architecture, agent architecture, programming language, or message-passing system.

• A UML (Agent Unified Modeling Language) is an evolving standard for a design methodology to support MAS. It is based on the UML methodology used with object oriented systems. This notation was proposed to adapt the UML’s one in order to describe the agent-oriented modeling [18].

AUML provides tools for:
- Specification protocol of interaction between agents,
- Representation of the internal behavior of an agent,
- Specification of roles, package interface agent, mobility, etc [2].

• The Agent Modeling Language (AML) is a semiformal visual modeling language for specifying, modeling and documenting systems that incorporate concepts drawn from multi-agents systems (MAS) theory [19].

• ASPECTS (Agent-oriented Software Process for Engineering Complex Systems) provides a holonic perspective to design MAS [20]. Considering that complex systems typically exhibit a hierarchical configuration, on the contrary to other methodologies, it uses holons instead of atomic entities. Holons, which are agents recursively composed by other agents, permit to design systems with different granularities until the requested tasks are manageable by individual entities.

The goal of the proposed meta-model of ASPECTS is to gather the advantages of organizational approaches as well as of those of the holonic vision in the modeling of complex system [1].

All these methodologies presented above are still quite recent. They are mainly focused on the analysis phase, whereas design and implementation phases are missing or are redirected to agent-oriented methodologies, which do not offer enough tools to model organizational concepts. Therefore, there is still a gap between analysis and design, which must be specified clearly, correctly and completely [14].

Finally, the maturity of methodologies can be analyzed by the number of systems that have adopted them. Most of analyzed methodologies have associated applications that show their feasibility. These methodologies have been applied in different fields such as medical informatics [21], manufacturing [20, 22], and e-commerce [23]. MaSE and INGENIAS are the most used ones. Unfortunately, the number of real world applications that use agent-oriented methodologies is still low [1].
4. THE MDA APPROACH
The MDA (Model Driven Architecture) proposes a methodological framework and architecture for systems development that focuses first on the functionality and application behavior, without worrying about the technology with which the application will be implemented. The implementation of the application goes through the transformation of business models in specific models to a target platform (Fig. 2). One research was done in this area as the dissertation of Jarraya T. [24]

Figure 2. The MDA approach [3]

The business process independent of automation, which comes from the expression of need, is described as a "CIM" (Computation Independent Model). The detailed functional analysis, the heart of the process is concentrated in the "PIM" (Platform Independent Model), which, as its name suggests, is strictly independent of the technical architecture and the target language.

The "PSM" (Platform Specific Model) is the model for engineering design obtained by transformation of PIM by projection on the target technical architecture. It is this model that is based on code generation [25].

The benefits to businesses on the MDA are primarily:
• The fact that architectures based on MDA are ready for technological developments.
• The ease of integrating applications and systems around a shared architecture.
• Broader interoperability for not being tied to a platform.

One of the main tools of MDA, we have AndroMDA who takes as its input a business model specified in the Unified Modeling Language (UML) and generates significant portions of the layers needed to build, for example, a Java application [26]. AndroMDA's ability to automatically translate high-level business specifications into production quality code results in significant time savings when implementing Java applications. The diagram below maps various application layers to, for examples, Java technologies supported by AndroMDA [25].

Figure 3. Application layers supported by AndroMDA [3]
The generation process of AndroMDA is as follows [25]:
• Preparation of the project in MagicDraw
• Preparation of use cases
• Preparation of class diagram
• Preparation of state charts
• Code Generation
• Generating the database
• Deploy the application

5. THE PROPOSED APPROACH
Our approach is based on model driven architecture (MDA), which aims to establish the link between the existing agent architectures and models or meta-model multi-agent systems that we build based on AUML. Our idea is to offer a design methodology based on agents AUML notation for establishing a generic class diagram that the designer can use to design his system [4]. This diagram is considered as a meta-model which is not generated by any tool and must be defined by the modeler himself.
Figure 5. AUML generic class diagram for a MAS

Our approach has a lot of benefits, it allows:
- Reducing costs and development times for new applications.
- Improving quality of applications.
- Reducing complexity of application development.
- Ability to generate all the necessary components described.
- Modularity and reusability of the developments.
- Coercion by the MDA model.
- Generating a library of generic models.

5.1 Description of the AUML generic class diagram
The diagram is conceived in three layers, each one is represented by a relationship between classes: A first part which is a relation between agent and its environment, a second part of specialisation of the agent class, and at the last part, a specialisation of the cognitive agent class [3].

5.1.1 The first part
The first part consists of two important classes:
- Environment
- Agent

- Environment is an important class on the diagram because it influences all the system. Environment’s data is represented by two sections, Attributes and Perceptions. Attributes can be considered as the all information that an environment should have, plus the following common information:
  - **Deterministic** when the next state of the environment is determined in a unique way by the current state and action of the agent, so the environment is deterministic. If the outcome is uncertain (especially if, as a result of the agent action, the environment can evolve in different ways), we are in the non-deterministic case.
  - **Static** if the environment cannot change its state without the intervention of the agent. The environment is dynamic if its state can change without the action of the agent in the time interval between two perceptions of the agent.
  - **Continuous** if any portion of an environment state to another requires passing through a sequence of intermediate states, otherwise the environment is discrete.

Perception is a section where the designer should determine all environment perceptions, example: number of agents.

Environment contains several functions allowing to start running, to perceive information from agents linked to it and to modify its state after each action from those agents, that is respectively Run(), Perceive() and ModifState().

- **Agent** is the main class on the diagram that allows the designer to express all agent properties. The constructor of Agents takes three sections: Roles, Attributes and Perception. Roles are agent functionalities. Attributes are all information that an agent should possess. And finally Perception which is a section where the designer should determine all agents’ perceptions about his environment or the other agents.

Agent contains several functions who allows starting running, perceiving information from environment or agents linked to it and executing all its actions, that is respectively Run(), Perceive() and Act().

The first part consists also of two important association classes:
- Action, between agent and his environment.
- Interaction, between agents.

- **Action** is an association class between agent and
environment. It lists all possible actions that an agent can execute on his environment.

- **Interaction** is a reflexive association class between agents. Agent can request information by the `getInformation()` function and send it by the `inform()` function. Agent may also deal with some constraints that it is possible to inform by the function `informaboutConstraints()`. The acceptance of partnership is added also to the main functionalities of Agent by the function `acceptPartnerShip()`.

### 5.1.2 The second part

The second part represents a specialisation relation of the Agent class. It consists of three important classes:

- Reactive agent,
- Cognitive agent,
- Communicative agent.

- **Reactive agent** (represented in green color) is a type of agent. It possesses the same properties of the Agent class.

- **Cognitive agent** (represented in clear blue color) is another specialization of the Agent class. In this class, the designer should determine the representations of the agent that he must have during its execution. The class possesses also one important function “Decide()” where agent can decide to execute an action or not according to his goals.

- **Communicative agent** (represented in orange color) is the last specialization of the Agent class. Like Cognitive agent class, Communicative agent class has representations but possesses a different function called “Communicate()” where agent must use to communicate his information to the other agents.

### 5.1.3 The third part

The third part represents a specialization relation of the Cognitive agent class. It consists of three important classes:

- Adaptive agent,
- Intentional agent,
- Rational agent.

- **Adaptive agent** (represented in pink color) is a type of cognitive agent. It possesses the same properties of the Agent class, the knowledge base and the “Decide()” function. As mentioned in the types of agent section above, an adaptive agent is able to change its objectives and its knowledge base as and when these changes. This functionality is expressed by the “Change_information()” function.

- **Intentional agent or BDI Agent** (represented in grey color) is designed from the "Belief-Desire-Intention" model. It is a type of cognitive agent. In the same case of Adaptive Agent class, this class possesses the same properties of the Agent class, the knowledge base and the “Decide()” function.

In this class, the designer should determine the agent’s beliefs represented by the Beliefs section. The beliefs of an agent are the information that the agent has on the environment and other agents that exist in the same environment. Beliefs may be incorrect, incomplete or uncertain, and because of that, they are different from knowledge of the agent, which is information still true. Beliefs can change over time as the agent by its ability to perceive or interact with other agents, collects more information.

The designer should also determine the agent’s intentions represented by the Intentions section. The intentions of an agent are the actions it has decided to do to accomplish his goals.

To choose the correct agent’s beliefs from the incorrect ones, this class offers the “Revise_beliefs(Pres, Belief)” function which is based on the agent’s knowledge base and his beliefs. Then, the “Generate_desires(Belief, int)” function comes to generate all the agent’s desires that he may be able to accomplish at once. The desires of an agent representing all things the agent would like to see made. An agent may have conflicting desires, in which case he must choose between her desires a subset that is consistent. This subset consists of his desires is identified with the beliefs and the intentions of the agent.

Another function comes after that, the “Filter(Belief, Generate_desires, int)” which filters all those elements above and gives the consistent beliefs, desires and intentions of the intentional agent.

Finally, the agent can select his actions according to this filtration and execute them by the “Actions_selection(Filter)” function.

- **Rational agent** (represented in dark blue color) is the last specialization of the Cognitive Agent class. Like Intentional Agent class, Rational Agent class has the Beliefs and the
Intentions sections but possesses just one function called “Mesure_performance(Percept, Belief)” where agent must use to execute his actions as efficient as possible. This function is based both on his perceptions and his beliefs.

5.2 The generic UML Class Diagram

This generic AUML class diagram was subsequently converted into a generic class diagram based on UML notation. This transformation will allow the designer to easily use AndroMDA to generate the source code equivalent to its UML diagram [1].

The translation from AUML to UML diagram is performed by following the steps below:
1. Keep the same titles of classes and associations which constitute the AUML diagram.
2. Assign roles, perceptions, intentions, beliefs and representations of each agent, and any possible additional attributes, in the attributes part of the UML class.
3. Combine all methods or functions in the operations part of the UML class.
4. Create a class for each representations, believes, desires and intentions sections.

We can obtain, in the end, the following result shown in Fig. 6:

Figure 6. UML generic class diagram for a MAS [3]

Our approach can present one disadvantage; it is the complexity of generating a good code source by AndroMDA. The model developed at the design phase, should be reliable in order to build the application and realize its implementation without errors [3]. This approach should be used by developers who have large and advanced notions of software programming.

6. APPLICATION EXAMPLE

6.1 Description

Our proposed AUML class diagram was used for design of one multi-agent system for a Collaborative information system based on agents whose a generic architecture was proposed on the thesis [27]. This architecture is based on the layers mentioned above:

We will now detail the different agents that manage the layers mentioned above:

- **The GUI agent**: This agent can be seen as a wizard that allows users to interact with the system. It allows acquiring all user requests, transforming in understood goals by the system, communicating these requests to the Manager agent and displaying the results to users. It also checks the identification of users (authentication) and sends their profiles to the Manager agent [28]. This agent is considered as a reactive agent (represented in green color in the generic diagram) due to its stimulus-action mode after receiving users’ requests and Manager Agent’s responses.

- **The Manager agent**: This agent is considered as the system mediator, the agent process the target set by the GUI agent and is able to reformulate this goal into sub-queries that are transferred to the MAS planning. When the sub queries execution is completed, the Manager agent collects the results found in the form of a comprehensive and global response [28]. This agent is considered as a cognitive agent (represented in clear blue color in the generic diagram) due to its capacity to remember all requests that he processed and its ability to choose the appropriate Composer Agent.

- **The Composer agent**: This agent represents a specified type of supply chain partners. There is a customer agent composer, a supplier agent composer, a carrier agent composer, etc... For each sub-request sent by the Manager Agent, one of the composer agent will performs a series of processing and proposes a solution plan. It communicates also with internal resources (databases / services base) and external resources (databases / remote services) to execute the established action plan.
This agent is considered as a reactive agent (represented in green color in the generic diagram) due to its stimulus-action mode after receiving Manager Agent’s requests and this by calling the appropriate service and by collecting Service Agent’s responses.

- **The Service agent:** This agent loads the necessary services to perform the action plan produced by one of the Composer Agent.

This agent is considered as a cognitive agent (represented in clear blue color in the generic diagram) due to its capacity to remember all action plans that he received from a Composer Agent and its ability to decide of the appropriate service to perform this action plan.

We can obtain the following AUML and UML diagrams corresponding to this example, shown in the Figures 7 below and 8 in the next page:

6.2 Realization

To validate our model for this example, we’ve tried to download AndroMDA with all the required dependencies (including all profiles referenced by models). Then, we generated our project « sysCol » by running « mvn org.andromda.maven.plugins:andromdaapp-maven-plugin:3.4-SNAPSHOT:generate ». The result of this command is as follows:

![Image of command output]

When we examine the various folders and files created by the andromdapp plug-in, we will notice files called pom.xml in various folders under sysCol. These files make up several Maven projects. In fact, the sysCol directory contains a hierarchy of Maven projects as shown below [26].

```
sysCol
|-- mda
|-- common
|-- core
|-- web
   |-- app
```

- **sysCol:** This is the master project that controls the overall build process and common properties.
- **mda:** The mda project is the most important sub-project of the application. It houses the sysCol UML model under the src/main/uml directory. The mda project is also where AndroMDA is configured to generate the files needed to assemble the application.
- **common:** The common sub-project collects resources and classes that are shared among other sub-projects. These include value objects and embedded values.

![Image of AUML Class diagram for a collaborative information system]
• **core**: The core sub-project collects resources and classes that use the Spring framework, optionally making use of Hibernate and/or EJBs under the hood. These include entity classes, data access objects, hibernate mapping files, and services.

• **web**: The web sub-project collects those resources and classes that make up the presentation layer.

• **app**: The app sub-project collects those resources and classes that are required to build the .ear bundle.

By opening the file “sysCol.xml” in MagicDraw, we will be able to build various graphs of our model to generate then the entire application source code. Note that AndroMDA can't read MagicDraw 17 models directly. Therefore, it can be exported to another file format: EMF-UML2.

After importing AndroMDA profiles to use for our application, we’ve designed our class diagram. The result of exporting of our “sysCol” model to EMF-UML2 format is located in the C:/sysCol/mda/src/main/uml folder. Below its content:

• **sysCol.xml**: the MagicDraw 17 model file.
• **sysCol.uml**: sysCol model in EMF/UML2 format. It's the file that will be processed by AndroMDA.
• 10 files ending with .profile.uml: the different profiles used by sysCol.uml.
Following our model definition, the application code
generation is achieved by executing the "mvn install"
command.

Thus, all classes corresponding to agents of our
collaborative system are created and their source codes
can be easily accessed and modified by the developer
where he has the ability to implement its operations in the
generated code.
Below, an example of the initial generated Manager
Agent source code before developer’s update:

```java
package java.code.Agent;
import java.code.Agent;
import java.util.*;
public class Manager extends Agent {
    public void Run() {
        // Agent initiating
        System.out.println("The Manager-agent "+getAID()+getName()+" is ready.");
    }
    public void Perceive() {
        // Agent perception
        // Get the title of the total request
        Object[] args = getArguments();
        if (args != null && args.length > 0) {
            String targetRequest = (String) args[0];
            System.out.println("Target request is "+targetRequest);
            ...
        }
    }
}
```

7. CONCLUSION

The purpose of this paper is to demonstrate the feasibility
of our approach to analyze, design and implement multi-
agent systems. With AUML modeling and MDA, we can
generate all the necessary components described by the
class meta-model that we proposed. Which leads us to
obtain a generic design based on SOA more or less
reusable components using one of the most MDA tools
used in development is AndroMDA [3].

In the future, we would like to model another application
sample containing adaptive or rational agents by using
another MDA tool like JadeMDA, etc. It will help us to
validate the efficacy of our proposed approach and lead us
to consider it as a generic approach which can be adopted
by every type of information system and used for any real
world application.

References
Generic Agent-oriented Model Used for Designing a
Collaborative Information System”, Proceedings of
the IEEE International Conference on Complex
Systems (ICCS), November 2012.
Agent Systems”, Proceedings of the IEEE
International Conference on Multimedia Computing
and Systems (ICMC), 2014.
Designing Multi-Agent Systems”, International
Journal of Advanced Computer Science and
systems, “A Modern Approach to Distributed
Artificial Intelligence”, England Massachusetts
Techniques for Agent-bases Systems”, Proceedings of
5th International Conference on Autonomous Agents,
Theory and Practice”, The Knowledge Engineering
Segrounichi, Intelligent agents, Agents Intelligents,
Politechnica University of Bucharest, 2002.
Gaia methodology: basic concepts and extensions”,
Methodologies and Software Engineering for Agent
Engineering with INGENIAS”, In Proceedings of the
international Central and Eastern European
Gómez-Rodríguez, J.C. González-Moreno, “A
technique for defining agent-oriented engineering
processes with tool support”, Journal of Artificial
Intelligence, vol.23, pp. 432-444.
INGLENIAS methodology and tools” in Agent-
oriented Methodologies, B. Henderson-Sellers and P.
development based on organizations”, Journal of
Electronic Notes in Theoretical Computer Science,
[15] IDK (INGLENIAS Development Kit), [Online].


**AUTHOR**

**Sara Maalal** was born in Rabat the Morocco’s capital in 1985. She received his professional master in Computer Engineering and Internet (3I), Security Networks and Systems Option, in 2008 from the Faculty of science of HASSAN II University, Casablanca, Morocco. In 2010 she joined the system architecture team of the National and High School of Electricity and Mechanic, Casablanca, Morocco. As a software engineer, her actual main research interests concern Designing and modeling Multi-Agent Systems.

**Wafaa Dachry** was born in Casablanca in 1985. She received her Ph.D. in Information System Engineering, in 2013, from the faculty of science of Hassan II University, Casablanca, Morocco. Currently, she is a Professor in information system, logistics ans electronic payment. Her actual main research interests concern information system applied to collaborative supply chain.

**Malika Addou** received her Ph.D. in Artificial Intelligence from University of Liege, Liege, Belgium, in 1992. She got her engineer degree in Computer Systems from the Mohammadia School of Engineers, Rabat, Morocco in 1982. She is Professor of Computer Science at the Hassania School of Public Works, Casablanca, since 1982. Her research focuses on Software Engineering (methods and technologies for design and development), on Information Systems (Distributed Systems) and on Artificial Intelligence (especially Multi-Agent Systems technologies).