Agent Oriented Software Engineering with INGENIAS[1]

Jorge J. Gómez-Sanz, Rubén Fuentes

Facultad de Informática,
Universidad Complutense de Madrid
{jjgomez,ruben}@sip.ucm.es

Abstract. INGENIAS is an agent oriented software engineering methodology for Multi-Agent Systems development. It combines agent research results with concepts and methods established in MESSAGE/UML. The result is a development process in the line of conventional software engineering processes, like object oriented software development paradigm or structured paradigm. INGENIAS defines deliverables and default activities to help in planning effort along a project. INGENIAS also provides with tools that facilitate the production of these deliverables. As a result, INGENIAS can be considered a valuable alternative in the Agent Oriented Software Engineering research field.

1. Introduction

Different proposals in the field of Agent Oriented Software Engineering (AOSE) try to integrate results from agent research with engineering practices, some from the perspective of agent theory (KAOS [1], Vowel Engineering [2], or GAIA [3]), some as an evolution of object-oriented systems (e.g. Kendall [4]), other as task execution models (e.g. Desire [5]), or from a knowledge-based systems approach (e.g. TROPOS [6] , MAS-CommonKADS [7]). Although they start from valid scientific assumptions, their application to real problems is still challenging because the only address specific phases of the software lifecycle, or concrete issues, such as goal extraction, definition and verification of communication protocols, or task execution. Another difficulty, so much important, is the scale of application of these methodologies and their ability to be incorporated in common software engineering practices. In most of the cases the engineer, the end user of the methodology, needs to acquire further expertise from what is required at his position in order to successfully apply the methodology.

Facing these problems, the ***grasia!*** (http://grasia.fdi.ucm.es) research group has developed a new methodology named INGENIAS. It relays on engineering concepts such as definition of workflows, encapsulation of functionality through roles, groups and organizations, or distinction of different views of the system, like structural or

[1] This work is sponsored in part by the Spanish Committee for Science and Technology (CICYT, TIC2000-0737-C03-02).

functional views. INGENIAS has been designed to build Multi-Agent System (MAS) specifications incrementally while ensuring correctness of the development and taking into account analysis, design, and implementation stages of the software development life cycle. The development effort invested in these stages is managed with the Unified Software Development Process (USDP) [8]. INGENIAS applies USDP to structure deliverables and activities involved in a MAS development.

INGENIAS reuses work from MESSAGE/UML, a methodology presented in AOSE'01 [9] and MAAMAW'01 [10]. MESSAGE/UML takes an approach close to engineering practices on dealing with MAS design. MESSAGE/UML defines five types of views of the system using a meta-modeling language. These views are applied along the development cycle to represent concrete aspects of the system. INGENIAS has improved original work from MESSAGE/UML by adding a new view (the environment view), reconstructing existing ones to adequate to the BDI model, augmenting its integration with the USDP, and providing tools to generate documentation of the system and to generate code automatically from the specification.

To prove its advantages, there are developments of public domain made with INGENIAS. These developments constitute an objective starting point for scientific discussion about the methodology. This is a remarkable difference with current trend in agent-oriented methodologies. Today, it is very hard to find complete case studies that show the suitability of an agent oriented methodology for an application domain. There are remarkable works, like DESIRE or TROPOS, which have been extensively documented and own dedicated conferences. However, results obtained are still far from the industry standards. Usually, demonstration of the suitability of a methodology is limited to examples that fit in a conference article, which is not close to the reality of software engineering.

This paper presents the core of INGENIAS and introduces some case studies that have been used for its development and validation. INGENIAS methodology is built around three elements: a visual specification language, development tools that support different stages of software development, and a software lifecycle paradigm. After presenting these elements, there is a brief introduction to case studies. Complete versions of these case studies are available through the web site of ***grasia!*** research group (http://grasia.fdi.ucm.es). Researchers can review them and discuss about if they do adequate to the problems addressed.

2. INGENIAS methodology

INGENIAS helps engineers to develop MAS using three elements:

- **A visual language for MAS definition.** This language has been created to allow working with MAS in a similar way as UML was created to work with Object Oriented systems. It summarizes results from existing agent research in form of a

hierarchy of concepts and relationships organized in views. Concepts, relationships, and views are defined with a meta-modeling language. The result is a meta-model which is easily extensible using primitives of the meta-modeling language. In the core work, we have applied GOPRR [11] as meta-modeling language, though this does not mean that other meta-modeling languages, like MOF [12], are automatically excluded. In fact, UML, which today is defined in terms of MOF primitives, can be described with GOPRR. We use GOPRR because at the time of INGENIAS development there was no meta-modeling tool that used MOF notation.

- **Integration with the development lifecycle of software development.** INGENIAS does not cover only one stage of the development process. There has been a significant effort in applying the methodology embedded in an industrial development process, the Unified Software Development Process. It defines a set of deliverables to be produced and a set of activities to produce them. These activities are distributed along USDP phases and workflows.
- **Development tools.** INGENIAS uses a meta-case tool (METAEDIT+ [11]) as an analysis/design environment. METAEDIT+ admits as input the visual language specification expressed with GOPRR and generates as output customized editors. These editors show to developers which elements they can use to describe a particular aspect of the system. From these editors, we have defined customized HTML documentation and special reports that are used as input for code generation tools (see section 5). The code generator itself is another tool developed for INGENIAS. It produces code according to the instructions given by the developer and some templates that will be filled with information from the specification of the system. With this tool is possible to generate code no matter what agent platform is being used.

These elements are detailed in following sections.

3. Defining MAS with INGENIAS

In INGENIAS, the general approach to specify MAS is to divide the problem in more concrete aspects that form different *views* of the system. This idea already appears in the work of Kendall [4], Vowel Engineering [2], MAS-CommonKADS [7], and later in GAIA [3]. The difference of this proposal from existing ones is how these views are defined and built, and how they are integrated in the MAS development.

Each type of view is described using a meta-modeling language, GOPRR [11] in our case. A meta-modeling language specifies a kind of grammars with which we build models, here known as views. Each one of these grammars is a meta-model and its instance a model. Meta-models describe what any MAS should have. Using these meta-models, engineers generate instances identifying entities that may appear in the future MAS. For each generic entity in the meta-model (e.g. meta-classes) the engineer must look for specific types of entities (e.g. classes) in the current problem

domain that share the features specified in the meta-model (relationships with other entities and related attributes). The final result is a view of the system under development, which is consistent with the high-level specification of what has to be in a MAS.

Meta-models are applied during the analysis when looking for a first version of the MAS. In this sense, a meta-model acts as a guideline for the analyst, by indicating which entities has to identify. During design, the meta-model is refined, by identifying new components and relationships among them, in order to achieve the appropriate level of detail. In parallel, the components of the meta-model are translated into computational entities, like state machines, session managers, or production rule engines. At the end of the process there is a specification of the MAS with a design that is based on concrete and implementable entities. As section 4 remarks, in each one of these stages, the developer has to produce deliverables that enable further work.

INGENIAS recognises five meta-models that describe system views. Each view, being the instance of a meta-model, is named *model*. So to describe a MAS, the developer must use the following models:

- **Agent model.** Describes single agents, their tasks, goals, initial mental state [13], and played roles. Moreover, agent models are used to describe intermediate states of agents. These states are presented using goals, facts, tasks, or any other system entity that helps in its state description. This way, an agent model could represent in what state should be an agent that starts a negotiation.
- **Interaction model**. Describes how interaction among agents takes place. Each interaction declaration includes involved actors, goals pursued by the interaction, and a description of the protocol that follows the interaction. There are several formalisms that can be used to describe an interaction. In INGENIAS, we have chosen UML collaboration diagrams and GRASIA[2] diagrams, however, there are no constraints in the nature of the protocol description formalism.
- **Tasks and goals model.** Describes relationships among goals and tasks, goal structures, and task structures. It is used also to express which are the inputs and outputs of the tasks and what are their effects on the environment or an agent's mental state.
- **Organization model.** Describes how system components (agents, roles, resources, and applications) are grouped together, which tasks are executed in common, which goals they share, and what constraints exists in the interaction among agents. These constraints are expressed in form of subordination and client-server relationships.

[2] GRASIA diagrams are experimental diagrams that include information of what ordering is applied to messages, how are delivered the messages, what is executed on receiving/sending a message, and what conditions are required to collaborate, initiate, or continue an interaction.

- **Environment model.** Defines agent's perception in terms of existing elements of the system. It also identifies system resources and who is responsible of their management.

A system will be composed of several models like the ones described previously. There are no constraints in the number of models of each type. As in UML, the end user will decide whether he needs one *tasks and goal model* or several.

The meta-models that describe these models are the result of the experience gained in several projects: *MESSAGE* (Eurescom P907 [14]), *Communications Management Process Integration Using Software Agents* (Eurescom P815 [15]), and *Personalised Services for Integrated Internet Information* (PSI3 [16]) . These meta-models define the architecture of the MAS (organization meta-model), key features of agents (agent meta-model), agent co-operation (interaction meta-model), and goals and tasks of the MAS (goal/task model). Information elements that represent domain specific information do not require the definition of a specific meta-model, as they can be modelled following a classical object-oriented methodology (i.e. a taxonomy of concepts with aggregation and inheritance relationships).

4. USDP with INGENIAS

To instantiate each meta-model, we have defined a set of activities (around seventy) in the software development process that lead to the final MAS specification. Initially, activities are organised in UML activity diagrams showing dependencies between them. Instead of presenting these activities here (a detailed description can be found in [17]), this section focus in what results are expected along the development. Fig. 1 summarises the results required in each phase of the USDP. These results are expressed using the visual specification language of INGENIAS the same way as UML is used to specify object oriented applications.

		PHASES		
		Inception	**Elaboration**	**Construction**
WORKFLOWS	**Analysis**	o Generate use cases and identify actions of these use cases with interaction models. o Sketch a system architecture with an organization model. o Generate enviroment models to represent results from requirement gathering stage	o Refined use cases o Agent models that detail elements of the system architecture. o Workflows and tasks in organization models o Models of tasks and goals to highlight control constraints (main goals, goal decomposition) o Refinements of environment model to include new environment elements	o Refinements on existing models to cover use cases
	Design	o Generate prototypes perhaps with rapid application development tool such as ZEUS o Agent Tool.	o Refinements in workflows o Interaction models that show how tasks are executed. o Models of tasks and goals that reflect dependencies and needs identified in workflows and how system goals are achieved o Agent models to show required mental state patterns	o Generate new models o Social relationships that perfect organization behaviour.

Fig. 1. Results to be obtained in each phase of the development process

In the analysis-inception phase, organization models are produced to sketch how the MAS looks like. This result, equivalent to a MAS architecture, is refined late in the analysis-elaboration phase to identify common goals of the agents and relevant tasks to be performed by each agent. Task execution has to be justified in terms of organizational or agent's goals (with task-goal models). This leads to identify the results that are needed to consider a goal as satisfied (or failed). In the design-elaboration phase, more detail is added, by defining workflows among the different agents (with organization models), completing workflow definition with agent interactions (with interaction models), and refining agent's mental state as a consequence (with agent models). According to the Unified Software Development Process, the goal of elaboration phase is to generate a stable architecture, so only the most significant use cases should be considered (the key functionality of the system). Remaining use cases, which are supposed to deal with special situations but that do not provide changes in the system architecture, are left to the construction phase.

We have used a meta-tool, METAEDIT+, that takes as input the meta-models specifications and generates graphical tools that are used by the developer to produce models using the concepts described in the meta-models (for which we have also associated a graphical representation). Models developed using these graphical tools consist of the agent concepts described in the meta-models. The examples accessible from ***grasia!*** web site shows how an specification generated with METAEDIT+ looks like.

5. Implementation with INGENIAS

The goal of implementation stage is to realize a system specification. Usually, it is a task manually executed by programmers, perhaps reusing code from existing agent platforms. However, this task can be also partially automated, as ZEUS [18] or agentTool [19] have demonstrated. These tools generate code mapping (or transforming) elements from the specification to concrete computational elements in the target framework. However, these tools are not prepared to support changes in the target framework (see comments in section 8).

INGENIAS considers implementation as a customizable procedure that maps system specification elements to concrete framework components. This procedure assumes that the target framework already exists. Examples of valid frameworks are JADE [20] or ZEUS [18]. Once key components have been identified in the framework, what remains is the problem of mapping specification elements to these concrete components.

Currently, the mapping process relays on two tasks:

- **Translate an INGENIAS specification to an intermediate format.** This format is expressed with the DTD from figure Fig. 2 (A). The developer has to define an extraction method that obtains instances of this DTD from an INGENIAS specification.
- **Define what will be replaced in the concrete components.** The developer has to mark up sources to convert them in documents compliant with the DTD from figure Fig. 2 (B). Though the format is similar to the one from previous task, both have different, though complementary, purposes.

```
<?xml version="1.0" encoding="UTF-
8"?>
<!ELEMENT repeat (#PCDATA | v |
repeat )*>
<!ATTLIST repeat
    id CDATA #REQUIRED>
<!ELEMENT sequences
(#PCDATA|repeat|v)*>
<!ELEMENT v (#PCDATA)>
<!ATTLIST v
    id CDATA #REQUIRED>
```

(A)

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT file (#PCDATA | v)*>
<!ATTLIST file
    overwrite (yes|no) #REQUIRED>
<!ELEMENT program
(#PCDATA|repeat|saveto|v)*>
<!ELEMENT repeat (#PCDATA | saveto | v)*>
<!ATTLIST repeat
    id CDATA #REQUIRED>
<!ELEMENT saveto (file, text)>
<!ELEMENT text (#PCDATA | repeat | v |
saveto)*>
<!ELEMENT v (#PCDATA)>
```

(B)

Fig. 2. (A) DTD for information extracted from the specification, (B) DTD for templates to be filled with the information extracted from the specification

To execute these tasks, currently there is no stable tool support. The developer must implement programs to extract information from an INGENIAS specification and mark up sources by hand.

Once these tasks have been performed, the code generation process can continue without human assistance. With documents of both types, Fig. 2 (A) and (B), it consists on replacing tags on source code with the information extracted from the specification.

5.1. Marking up sources

The mark up of sources bases on the hypothesis that many coding work consists on copying and pasting pieces of code that require minor modifications to provide a concrete functionality. We may say that these pieces of code constitute a *pattern of code*. A *pattern of code* stands for a piece of code that can be reused for different elements of the implementation, like *set* and *get* methods to access object's attributes (see the example in Fig. 3). DTD from Fig. 2 (B) allows to define these patterns.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE program SYSTEM "template.dtd">
<program>
     <repeat id="informationEntities">
          <saveto>
               <file overwrite="yes">
/home/developer/gen/codegen/output/masrobocode/knowledge/<v>ieid</v>IE.java
</file>
               <text>

package masrobocode.knowledge;

public class <v>ieid</v>IE extends masrobocode.knowledge.InformationEntity{

   <repeat id="slots">
                    <v>slottype</v>
                    <v>slotname</v>;
   </repeat>

   public <v>ieid</v>IE(<repeat id="params">
                    <v>slottype</v>
                    <v>slotname</v>
                </repeat>) {
     <repeat id="initialize">
     this.<v>slotname</v>=<v>slotname</v>;
     </repeat>
   }

   <repeat id="methods">
    public void set<v>slotname</v>(<v>slottype</v>
                    <v>slotname</v>){
     this.<v>slotname</v>=<v>slotname</v>;
     };
    public  <v>slottype</v> get<v>slotname</v>(){
     return <v>slotname</v>;
     };
   </repeat>

}
</text>
          </saveto>
     </repeat>
</program>
```

Fig. 3 Template for Information Entity using DTD from Fig. 2 (B)

These patterns are instantiated with documents compliant with DTD from Fig. 2 (A). These documents have as root a *sequences* tag. A *sequence* is built of *repeat* and *v* tags. The semantic of both tags is associated with the markup of sources. The idea is to replace *v* (a variable) tags from sources by values associated with *v* tags extracted from the specification, and *repeat* tags with duplicates of the code among the *repeat* begin and end tags. The extra tag *saveto* is used to save pieces of code, allocated among *text* tags, to concrete files, identified by *file* tags.

With such simple markup language, it is possible to generate entities like the one shown in Fig. 3. Fig. 3 presents a template for *facts* and *events* of the **robot battles** example (see section 6.5). In the specification, a fact or an event is an entity with slots. A valid representation for these elements in the implementation is a Java object whose attributes are the slots of the fact or event.

Developers define similar mappings for other elements. For example, goal satisfaction conditions are mapped to the left hand side of production rules that process the output of tasks. And task execution is associated with the right hand side of rules that represents decision mechanisms of the agent. At the end, there are sets of mappings that generate the final system when applied together.

5.2. Code generation tool

Existing mappings are codified in the code generation tool as shows Fig. 4 and Fig. 5.

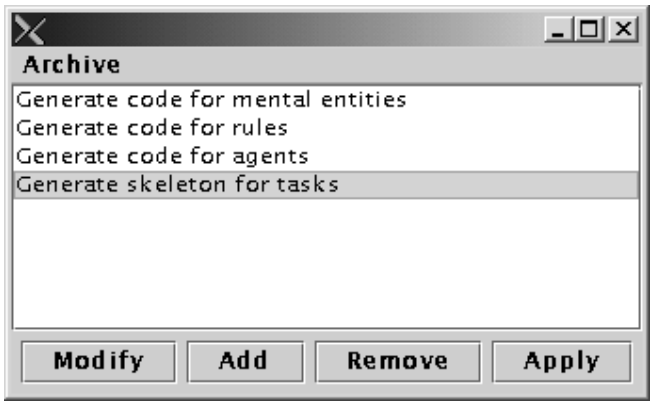

Fig. 4. Application that handles known mappings from system specification to code templates. In the picture, code generation for tasks is selected.

Fig. 4 presents the application that handles mappings for the **robot battles** example (see section 6.5). From this application, we apply each mapping to obtain sets of Java classes and ILOG JRules files that form the MAS.

Each mapping (see Fig. 5) is specified with a template file, like the one of Fig. 3, a transformation program that extracts information from an INGENIAS specification (according to the DTD of Fig. 2 (A)), a target which is the name assigned to the output file, and selected models on which the transformation will be applied. Additionally, the developer can define a program responsible of deciding whether the transformation is possible or not.

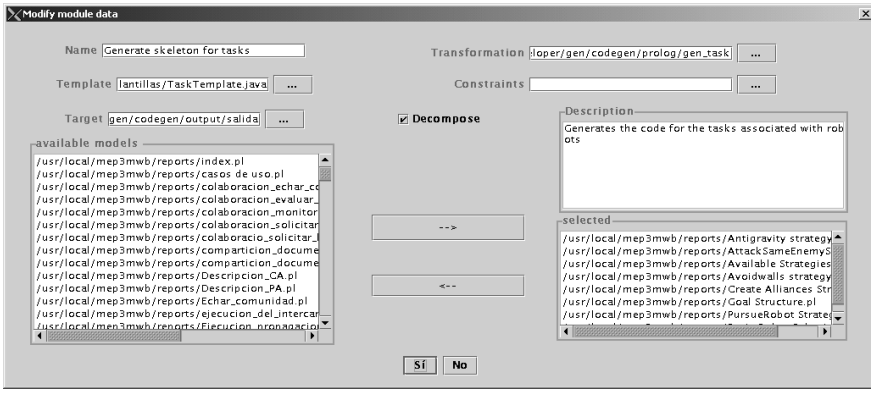

Fig. 5. Definition of a mapping for task code generation. This windows appears of selecting *modify* in the window from Fig. 4

This kind of code generation depends strongly on the target platform and translates the problem of writing code to how the design fits into a concrete, already existing, agent based framework. However, the process itself is very simple and easily customizable. The main obstacle at the moment is the interface with METAEDIT+, since it does not provide direct methods to access the specification from Java or any other language. Instead, developers have to generate a pre-defined report that translates each diagram into a PROLOG predicate. Starting from these predicates, we define several rules to produce documents compliant with DTD from Fig. 2 (A).

At the moment, code generation involves an important human intervention at the beginning. However, we expect to reduce it significantly on having direct access to the diagrams. On the other hand, the benefits are remarkable, since, when the code generation tasks have finished, with one click, we can update source code of the whole system keeping consistency with original specification.

6. Testing the methodology

INGENIAS has been tested in three case studies. These are available in the ***grasia!*** web-site (http://grasia.fdi.ucm.es). Due their length, this section provides only a brief explanation of what problems addresses each one. There exists an operational prototype of the first, and on-going developments of the other two.

6.3. Collaborative Information Filtering

Given the amount of documents currently available in the web, services for recommending documents and web sites of interest for particular users are a useful tool for improving the access to relevant information. Document recommendation tools can be offered directly as a service, or can be used to support the provision of added-value services, normally involving a high degree of personalization. The case study aims at developing new functions and services to enhance the existing methods and tools for information management and retrieval on Internet, with focus on personalization of the information delivered to the users, and builds these tools on a multi-agent system. This paper presents how this agent system is used to support a document recommendation service, which is currently working in three different commercial web sites.

Document recommendation tools usually rely on text mining algorithms that analyze a set of documents. Although some of these systems, which are reviewed in the next section, have got interesting results, we consider that the user is also able to collaborate by providing subjective information that, in combination with analytical methods, can improve the quality of the information resulting from the recommendation process. In order to integrate users in the recommendation system with minimum annoyance for them, personal agents representing the users in the system can alleviate the interactions with them, and play user's role in the document recommendation system workflow. The workflow defines the activities and information exchange among the parties in the system. Apart from personal agents, community agents are defined, which wrap the access to text mining tools, reduce the number of interactions required among personal agents, and define politics for information broadcast.

The result is a collaborative filtering system with interesting properties from a web engineering viewpoint: scalability (to support increasing number of requests), robustness (to improve system availability), and flexibility (agents are used to model circumstances that may arise on real web sites, like work load, behavior of single users, and behavior of the community of users).

The system described assumes that users are already subscribed to a group. The aim is to ensure that shared documents follow certain patterns. The kind of information distributed should be acceptable by members. The interests of the users change along the time. What cannot be tested here is the appropriateness of the underlying categorization system against humans. This is certainly a frontier to overcome, but agent technology cannot provide final solutions since it is a problem of pure user modeling. However, we can abstract from the conceptual tools used to model the user and conceive it as a categorization problem: given a new document can it be classified in the category represented by the user. Agents use this function to know if a new document may be of interest to the user or not. Wrapping categorization tools as agents that can interact with other agents in the system workflow is possible to provide mixed information filtering solutions (autonomous filtering with categorization tools and manual filtering by users).

6.4. Agents for Personal Computer Management

Managing a personal computer is a never-ending task. Users have to install programs, remove those less frequently used to free memory and hard disk, and organize the information they require or produce. Operating systems provide a limited support to manage with these programs. This aid concerns mainly with their installation and removal, avoiding collisions in the access to system resources and shared libraries. However, there is no aid in organizing existing documents in a PC, perhaps due to the heterogeneity of the format of documents. Fortunately, today there are free tools able to understand different formats, and enabling a better management of users' documents. In this example, we are going to construct a MAS to aid users in managing documents stored hard disk or e-mail accounts.

According to the research work in dealing with this information overload [21] [22], we can construct agents that may save effort to the user by acquiring his preferences (e.g. about categorizing, naming or formats) and fulfilling high level orders (e.g. move text documents to the appropiate folders or delete spam e-mail). The performance of these agents should increase with time: the most the user works with the system, the most the system knows about its user and therefore the better it behaves.

The design of these agents should take into account that there are many programs in a PC and that their number varies with user needs. Each of these programs is a candidate information source of documents. To handle the output of these programs will require further system modifications. Ideally it should be done without disrupting the work of the user.

The goal is to construct a scalable system that allows the user to manage his documents with high-level features and without caring who produces these documents. In this development, the system works only with e-mail and documents in the hard disk. The system supports actions like categorizing documents and moving them to appropriate folders according user's preferences. A requirement is that the process has to be accomplished minimizing user's intervention.

6.5. Robot battles

Many AI researchers find out that games are excellent test-bets for their theories. The advantage of games is that there exists an environment and a concrete interface to interact with it. For example, there are applications of SOAR [23] architecture to control a bot inside a quake™ game to play against other bots or humans. In the quake™ game, the bot can hear the steps of other players and see them. This way, constructing a bot in quake™ becomes a simplified version of constructing a robot in a real environment.

Following this research trend, this example shows how to acquire elements from a game to express them in the methodology language. The chosen game for this

example is Robocode. Robocode is a Java game distributed by IBM whose aim is to teach Java programming in an environment that motivates learning. The game is about tank-shaped robots fighting among themselves. Each robot can actuate with predefined primitives (like ahead, turn left, turn right) and perceive the environment with radar, position sensors and some events. Each robot tries to survive the battle with different strategies (some of them are commented in Robocode tutorial pages). Java programmers have to figure out, without any human intervention, which sequence of actions will improve the chances that a robot survives.

As an addition, our example shows how cooperation among robots may take place. Original Robocode did not allow this, so to test the example it was required to modify its source code to allow communication among different robots. The main threat here was to connect INGENIAS with Robocode at the conceptual level, studying with INGENIAS the analysis and design of a Robocode system, and at the code level, studying how these ideas may be reflected in form of real Java classes.

From the specification, now it is possible to generate a set of skeleton classes. These classes can be regenerated anytime so that they always stick to the specification.

7. What is missing in INGENIAS?

The experience with INGENIAS so far has discovered some open issues:

- **Primitives to express algorithms.** INGENIAS encapsulates algorithms in tasks and describes only its inputs and outputs. To express algorithms directly, there are two alternatives which are being considered: creating custom primitives to express algorithms or linking with better methods to express algorithms, like UML sequence diagrams or UML activity diagrams.
- **Explicit representation of deployments.** Being integrated with RUP, INGENIAS can reuse RUP deployment specifications. This way, deployments diagrams makes possible to associate specification elements with implementation ones and determine how many instances there will exist of each element. Currently, there is no support for deployment diagrams in current version of INGENIAS. Instead, INGENIAS delegates this task to existing UML compliant development tools.
- **Management of graphs.** The number of diagrams produced in current experiments is high. METAEDIT+ provides grouping by type, but this is not enough. In UML the solution is to define diagrams within packages. With INGENIAS current support tool this is not possible. As temporal solution, documentation generation tools provide a navigable version of what is being specified in METAEDIT+. This version is easier to review. The future stable solution is under development: a new Java-based environment that supports INGENIAS visual language.
- **Code generation.** Though there is a generalized method to generate code from the specification, there are still gaps in the process. Developers have to participate actively customizing how marked-up sources are rewritten using the specification

generated with INGENIAS visual language. This is annoying, concretely when the developer has never developed a MAS and does not know in advance which agent architecture he will use or how an agent platform works.

- **Validation tools.** Though METAEDIT+ ensures that the meta-model is satisfied, there are still some semantic verification work that remains to be done. Like older specifications of UML, we have chosen natural language to express the semantic of INGENIAS visual specification language. This is an advantage and a drawback. It is an advantage because it adds flexibility to the analysis and the design. Ambiguity is useful in UML in the association relationships, because users can enrich the meaning of an association in a design so that it fits in his point of view of the system. The less ambiguous is the language, the more difficult is to express something. That is the reason why developers first think about the system in UML and then go to the implementation language, which is far less ambiguous. On the other hand, the more ambiguous is the language, the more difficult is to say if what it is being expressed is correct or not. But this is even more difficult, since can anybody define a general procedure that given an entity-relationship diagram representing a database says that the dababase is valid? Validation requires the existence of formal models that determine what is right or wrong.

8. INGENIAS and MaSE

From existing methodologies, the closest to what INGENIAS intends to do is MaSE. MaSE is a methodology documented in several research papers [19;24;25], development manuals, as well as PhD. thesis. The quality of MaSE is availed by a development tool that supports analysis and design as well as code generation. MaSE also provides a development process to guide developers. Despite its achievements, its orientation is different from INGENIAS. Following, we provide four key differences.

First, MaSE's approach centers mainly in applying object oriented techniques from UML [26] and misses important concepts highlighted in agent research, like planning or cognitive capabilities. INGENIAS starts from agent research results to build its visual specification language. In INGENIAS is possible to include in the development elements like plans (represented as workflows) or tasks that satisfy goals like in the BDI model [27], mental state of the agents (represented as an aggregation of mental entities like facts, events, or goals) and mental state patterns like in [13], group agents in organizations like in Ferber [28], or represent interactions as a combination of protocols like in GAIA [3].

Second, MaSE is strongly constrained by its development tool, *agentTool*. In INGENIAS, the visual language that defines the MAS is open to improvements without a strong development effort. Changes in MaSE may need further reengineering of agentTool, what constraints the evolution capabilities of MaSE. This is important, since a methodology requires years of experimentations and modifications until reaching stability.

Third, with respect code generation, INGENIAS bases on an open process of code generation, isolated from the analysis/design tool, and customizable by developers. AgentTool provides predefined code generation methods. However, these cannot be modified without rebuilding the tool. INGENIAS code generation method is more flexible. It relays on an XML-based markup language whose interpretation leads to the final code. Developers can modify sources and regenerate the code with no additional cost.

And fourth, the software development lifecycle in MaSE is a step-based process close to the waterfall paradigm [29]. Waterfall models have been deprecated today in medium-big size developments because they do not tolerate going-back actions. What happens if at the design stage the developer discovers that some functionality is not properly specified at the analysis? It perhaps may be a problem of a bad analysis, however, it happens. A solution is to apply an iterative process, like the RUP. An iterative process assumes that a development is not a sequence of steps but an iteration of activities that construct the final system incrementally. Instead of defining a customized development process, INGENIAS has been adapted to an existing and reliable one, the previously mentioned RUP. Also, INGENIAS visual specification language provides primitives with different level of abstraction. This feature makes possible to increase the level of detail from one iteration to another.

9. Conclusions

This paper has presented the core of INGENIAS and introduced some case studies used for its development. The complete case studies are available through the web site of GRASIA research group (http://grasia.fdi.ucm.es). Some of the results presented here briefly, have been described with detail in two papers. The core language was published in the SAC ACM 2002 [30] and its integration with software development process in Iberamia 2002 [31].

The most remarkable of INGENIAS is that it does not base in ideas of what a methodology should look like but on expertise in MAS development and on application to real cases. INGENIAS has been tested during one year, and now it is stable enough to be disseminated. Today, members of GRASIA and other volunteers are using it in research projects.

As section 7 has remarked, there are open lines of research, like obtaining formal representations of the semantics of the models, or create customized environments with tools for validation, analysis, design and implementation of MAS. However, current results are promising.

10. References

[1] Bradshaw, J. M. KAoS: An Open Agent Architecture Supporting Reuse, Interoperabiliby, and Extensibility. 1996. Knowledge Acquisition Workshop (KAW).

[2] Demazeau, Y. From cognitive interactions to collective behaviour in agent-based systems. 1995. European Conference on Cognitive Science.

[3] Wooldridge, M., Jennings, N. R., and Kinny, D., The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents and Multi-Agent Systems, vol. 15 2000.

[4] Kendall, E. A Methodology for Developing Agent Based Systems for Enterprise Integration. 1995. IFIP Working Conference of TC5 Special Interest Group on Architectures for Enterprise Integration.

[5] Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., and Treur, J., DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework, International Journal of Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, 1997.

[6] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. 2001. Proceedings of the fifth international conference on Autonomous agents, ACM.

[7] Iglesias, C., Mercedes Garijo, M., Gonzalez, J. C., and Velasco, J. R., Analysis and design of multiagent systems using MAS-CommonKADS, in Singh, M. P., Rao, A., and Wooldridge, M. J. (eds.) Intelligent Agents IV LNAI Volume 1365 ed. SpringerVerlag: Berlin, 1998.

[8] Jacobson, I., Rumbaugh, J., and Booch, G., The Unified Software Development Process Addison-Wesley, 1999.

[9] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J. J., Pavon, J., Kerney, P., Stark, J., and Massonet, P. Agent Oriented Analysis using MESSAGE/UML. Wooldridge, M., Weiss, G., and Cianciarini, P. Agent-Oriented Software Engineering II. 2001. Springer Verlag. LNCS 2222.

[10] Garijo, F., Gomez-Sanz, J. J., Pavon, J., and Massonet, P. Multi-Agent System Organization. An Engineering Perspective. 2001. MAAMAW 2001, Springer Verlag.

[11] Lyytinen, K. S. and Rossi, M. METAEDIT+ --- A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. LGNS#1080. 1999. Springer-Verlag.

[12] OMG. MOF. Meta Object Facility (specification). V.1.3. formal. 3-4-2000.

[13] Shoham, Y., Agent Oriented Programming, Artificial Intelligence, vol. 60 pp. 51-92, 1993.

[14] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J. J., Pavon, J., Kerney, P., Stark, J., and Massonet, P. Eurescom P907: MESSAGE - Methodology for Engineering Systems of Software Agents. http://www.eurescom.de/public/projects/P900-series/p907/default.asp . 2002.

[15] Eurescom P815. Communications Management Process Integration Using Software Agents. http://www.eurescom.de/public/projects/P800-series/p815/default.asp . 1999.

[16] PSI3. Personalized Service Integration Using Software Agents. IST-1999-11056. http://www.psi3.org/index.htm . 2001.

[17] Gomez-Sanz, J. J., Modelado de Sistemas Multi-Agente. Dpto. Sistemas Informáticos y Programación, Facultad de Informática, Universidad Complutense, 2002.

[18] Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C., ZEUS: A Toolkit for Building Distributed Multi-Agent Systems, Applied Artificial Intelligence Journal, vol. 1, no. 13, pp. 129-185, 1999.

[19] Wood, M. and DeLoach, S. Developing Multiagent Systems with agentTool. 2000. ATAL 2000. LNAI 1986. Castelfranchi, C. and Lespérance, Y.

[20] Bellifemine, F., Poggi, A., and Rimassa, G. JADE: a FIPA2000 compliant agent development environment. 2001. Proceedings of the fifth international conference on Autonomous agents, ACM.
[21] Maes, P., Agents that Reduces Work and Information Overload., Readings in Intelligent User Interfaces Morgan Kauffman Publishers, 1998.
[22] Lieberman, H. Letizia: An Agent That Assists Web Browsing. 1995. International Joint Conference on Artificial Intelligence.
[23] Laird, J. E., Newell, A., and Rosenbloom, P. S. SOAR: an architecture for general intelligence. Artificial Intelligence 33[1], 1-64. 1987.
[24] DeLoach, S. Analysis and Design using MaSE and agentTool. 2001. Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conferece (MAICS).
[25] DeLoach, S. A., Wood, M., and Sparkman, C. H., Multiagent Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering, vol. 11, no. 3, June2001.
[26] OMG. Unified Modeling Language Specification. Version 1.3. http://www.omg.org . 2000.
[27] Kinny, D., Georgeff, M., and Rao, A. A Methodology and Modelling Technique for Systems of BDI Agents. 1997. Australian Artificial Intelligence Institute.
[28] Ferber, J. and Gutknecht, O. A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. 1998. Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98), IEEE CS Press.
[29] Pressman, R. S., Software Engineering: A Practitioner's Approach New York: 1982.
[30] Gomez-Sanz, J. J., Pavon, J., and Garijo, F. Meta-modelling of Multi-Agent Systems. 2002. ACM. SAC 2002 ACM.
[31] Gomez-Sanz, J. J. and Pavon, J. Meta-modelling in Agent Oriented Software Engineering. 2002. IBERAMIA 2002 (to be published).