Advanced Studies in Artificial Intelligence
© 2014 Pushpa Publishing House, Allahabad, India
Published Online: March 2014
Available online at http://www.pphmj.com/journals/asai.htm
Volume 1, Number 1, 2014, Pages 27-44

TOWARDS TEACHING ARTIFICIAL INTELLIGENCE USING A MODEL-DRIVEN APPROACH

M. Gutierrez, J. Roa, W. Santana and G. Stegmayer

CIDISI-UTN-FRSF, CONICET, Lavaise 610-3000, Santa Fe, Argentina

Abstract: In computer science, it is quite difficult to teach well an introductory course on AI, partly because AI lacks a unified methodology, overlaps with many other disciplines, and involves a wide range of skills from very applied to quite formal. When teaching Artificial Intelligence, models are the principal artifacts used by professors to communicate concepts. These models should be used as part of a technical answer of a practical work, with the aim of narrowing the distance between concepts and their implementation in a programming language. The model-driven development presents a very promising approach to reduce the difficulties in the generation of code solutions. This work presents an MDD-based method, a language to define intelligent agent and a computational tool that gives support to the MDD-based method.

Keywords and phrases: artificial intelligence, education, intelligent agents, model-driven development.

1. Introduction

Nowadays, teaching Artificial Intelligence (AI) into systems engineering or computer science programs is a required subject. Undergraduate courses of AI cover a wide diversity of topics such as problem solving, knowledge representation, inference systems and machine learning among others. The AI course developed at Universidad Tecnológica Nacional - Facultad Regional Santa Fe, Argentina (UTN-FRSF) focuses on teaching software agents. The course adopts the definition of software agent proposed by Russell and Norvig in their traditional AI book [1], where an agent is everything that perceives its environment through sensors and responds or acts upon the environment through actuators.

Received: May 29, 2013; Accepted: August 1, 2013
Communicated by Peng-Yeng Yin

In the AI course, professors propose a problem as a practical work. Students must design and develop a software agent that solves the given problem. To this aim, students must take a decision about which learned artifact to use for agent developing. These artifacts may include different types of agents, strategies, techniques and algorithms. For instance, an agent may use search [1] as its main problem resolution strategy, or it may use situation calculus [1] or a planning technique [1]. The use of different techniques helps students to make a comparison in terms of performance and adequacy of the proposed answer.

Professors should guide students in the definition of agents and in the evaluation of the different strategies an agent may use as its main decision mechanism.

However, professors must face several challenges in the education of AI. It has been pointed out that in computer science it is quite difficult to teach well an introductory course on AI, partly because AI lacks a unified methodology, overlaps with many other disciplines, and involves a wide range of skills from very applied to quite formal [2]. There are three levels of abstraction to take into account: (i) conceptual modeling, expressing concepts and structures; (ii) program description, which focuses on aspects such as visibility, modularity, encapsulation; and (iii) implementation code, which focuses on programming and execution [3]. These different levels are usually time-consuming and entail intensive programming tasks. In addition, they require students to focus on the implementation details, which are not relevant to the conceptual problem itself, instead of directing their attention to the fundamental concepts of agents. Hence, when designing an AI course, the teacher must decide how much time, focus and effort are given to each of these different levels. A good balance among them is needed [3].

There are two important issues to take into account when teaching AI. On the one hand, teachers need to make the relations between theory, model and design of artifacts [4], and to narrow the gap between the concepts taught in class and the practices developed in the laboratory.

On the other hand, teachers need to take into account that students have already gone through different stages in which they have had to work and learn different techniques, paradigms, programming languages, algorithms, etc. Students must learn to design solutions based on the problem domain from a conceptual point of view and to exercise their abstraction capabilities in solving engineering problems, which is an essential skill that every information systems engineer should have.

For these reasons, we consider important the use of model-driven development (MDD) approaches [5] in AI teaching, so that students focus specifically on modeling the problem and learning the basic concepts of AI, leaving out some implementation details that may lead students to deviate from learning the AI concepts. The MDD approach is a recent technique of software engineering in which models are the main design artifacts and form the primary basis for generating (semi) automatically specifications of executable code. The MDD approach has been identified as a source of benefits in different areas of software engineering [6-9]. This paradigm shift has impacted not only the way how to implement software systems, but it is also changing the way software engineering is being taught [3, 10, 11]. In particular, we consider that the use of this approach in AI teaching will reduce the complexity and cut-down the time necessary to develop a practical work, while allows students focus on concepts.

Therefore, in this work, we propose an integrated solution for teaching AI composed of a method, a language and a tool.

The Model-Driven Teaching of Artificial Intelligence (MDT-AI) method is an MDD-based method to teach AI. To support the method, we defined the Agent Conceptual Modeling Language (ACML), which enables the definition of agent models using the concepts taught in the AI subject. In addition, we developed an Integrated Development Environment for Modeling and Executing Intelligent Agents (IDEM-IA) that supports the MDT-AI method. IDEM-IA provides functionalities to support the development of agents from a conceptual point of view through a graphical editor and a set of model-to-code transformations for the automatic generation of source code from agent models.

This work is structured as follows. Section 2 describes the MDT-AI method and the ACML language. Section 3 presents the tool IDEM-IA. Section 4 shows an example of its use in class. Section 5 presents an evaluation of the method. Finally, Section 6 presents the conclusions and future work.

2. Model-Driven Teaching of Artificial Intelligence

A key task in the teaching of AI is to make students learn the conceptual elements of software agents and how these elements relate with each other. Working with conceptual models of software agents is a key point to reach this task. To this aim, we defined the Model-Driven Teaching of Artificial Intelligence (MDT-AI) method. MDT-AI enables students to separate the conceptual definition of an agent such as its state, strategy, action, perception and environment, from its technological specification in a given programming language.

The development process of the MDT-AI method consists of two phases: *conceptual modeling of software agents* and *generation of agent specifications*. Following, we describe these phases.

2.1. Conceptual modeling of software agents

In order to define conceptual agent models, we defined the Agent Conceptual Modeling Language (ACML). This language was specifically defined to represent the concepts taught in the AI subject so as to facilitate the students learning. ACML allows students to seamlessly represent the definition of the components of a software agent using the concepts learned in class. Through the use of this language, it is possible to narrow the gap between the concepts taught in class and the practices developed in the laboratory.

Figure 1 shows the metamodel of ACML. This metamodel defines the domain of agents with a simple vocabulary like the one used in class. An *Agent* is associated with a *Goal*, a *Strategy* and a *State*. An *Agent* receives a set of *Perceptions* from the *Environment* and executes a set of *Actions* on such *Environment*. The *Environment* represents the thing with which *Agent* interacts. Both the *Agent* and the *Environment* have a given *State* that is represented through a *DataStructure*. All these elements are a part of the *AgentModel*, which represents the complete solution.

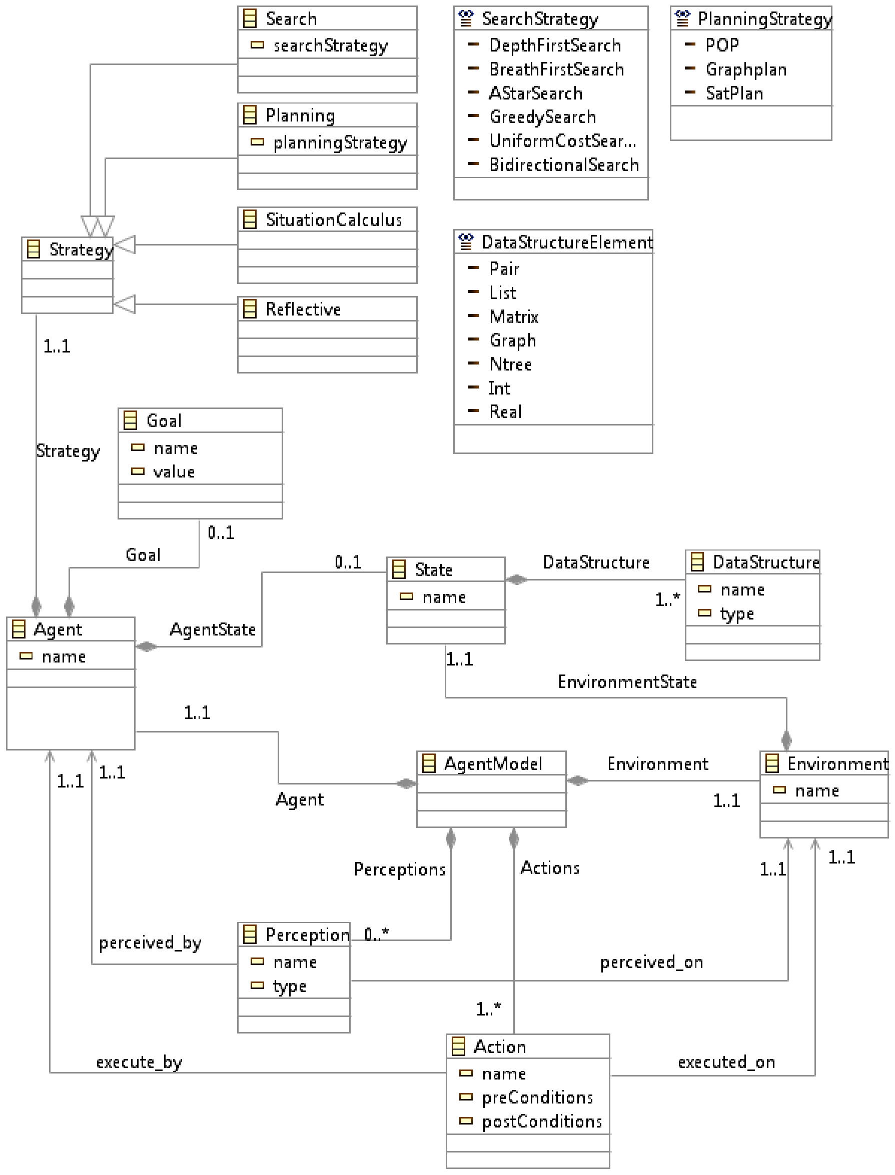


Figure 1. ACML metamodel.

The *Agent* uses a *Strategy* which defines the AI strategy used by agent to make a decision. There are four strategies: *Search*, *Planning*, *SituationCalculus* and *Reactive*. Search can be split into *DepthFirstSearch*, *BreathFirstSearch*, *AStarSearch*, *GreedySearch*, *UniformCostSearch* or *BidirectionalSearch*, each one representing different non-informed and informed search algorithms according to [1].

2.2. Generation of agent specifications

This phase entails the generation of agent specifications in a given programming language. Figure 2 shows the technology-independent transformation pattern to be used in the generation of executable code. To this aim, the pattern is composed of a set of model-to-code transformation rules. Basically, the rules take an ACML model representing an agent and its environment and generate the technology-dependent code which can then be executed to validate the solution proposed by students.

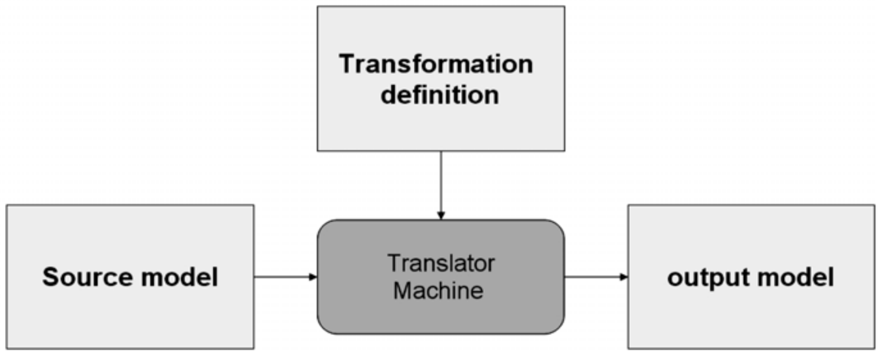


Figure 2. Technology-independent transformation pattern.

3. Tool Support

In order to support the MDT-AI method, we developed the Integrated Development Environment for Modeling and Implementing Software Agents (IDEM-IA). IDEM-IA[1] supports both the conceptual modeling of software

[1]http://code.google.com/p/idemia

agents as well as the generation of executable code. The generated code is based on FAIA[2], which is a framework for developing intelligent agents. Following, we describe both IDEM-IA and FAIA tools.

3.1. Integrated Development Environment for Modeling and Implementing Software Agents (IDEM-IA)

IDEM-IA was developed with the aim of providing a pedagogical tool to seamlessly guide students in an intuitive way in the development of software agents taking into account the following requirements: (1) support the definition of conceptual models of software agents with the concepts taught in class; (2) support the generation of Java[3] code from conceptual models of agents making use of the extension points of the FAIA framework; (3) provide extension mechanisms to add new AI strategies; (4) allow the addition of new functionalities and transformation machines to generate executable code in other programming languages. To support these requirements, the tool is based on the Eclipse platform [12] since this platform is widely used by software developers, it is extensible, and open source.

Figure 3 shows the architecture of IDEM-IA. The lower layer is the Eclipse platform. This platform is extensible by means of the addition of new modules known as plug-ins, which provide new functionality. The Eclipse Modeling Framework (EMF) [13] layer supports the definition of the metamodel of ACML, and provides to the upper layers the functionality necessary to manipulate the model instances of the ACML language. In addition, EMF is the basis to provide interoperability with other tools and applications of the Eclipse platform.

The following layer is composed of three frameworks: GMF, JET, and FAIA. The Graphical Modeling Framework (GMF) [14] is used to generate graphical editors. It provides the tools necessary for the generation of graphical components and the execution infrastructure necessary for the

[2]http://code.google.com/p/faia
[3]http://www.java.com

generation of the graphical editor of the ACML language. The Java Emitter Templates framework (JET) [15] provides the support to generate Java code from a conceptual model of an agent. The generated Java code makes use of the extension points provided by the FAIA framework.

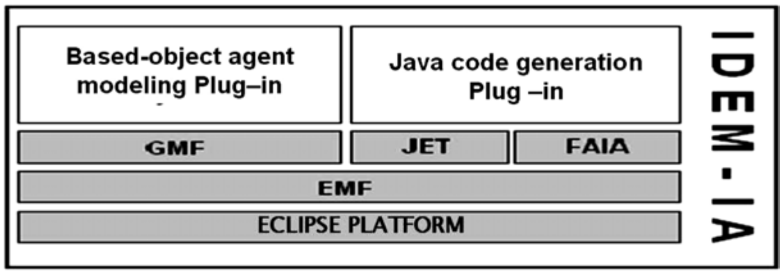
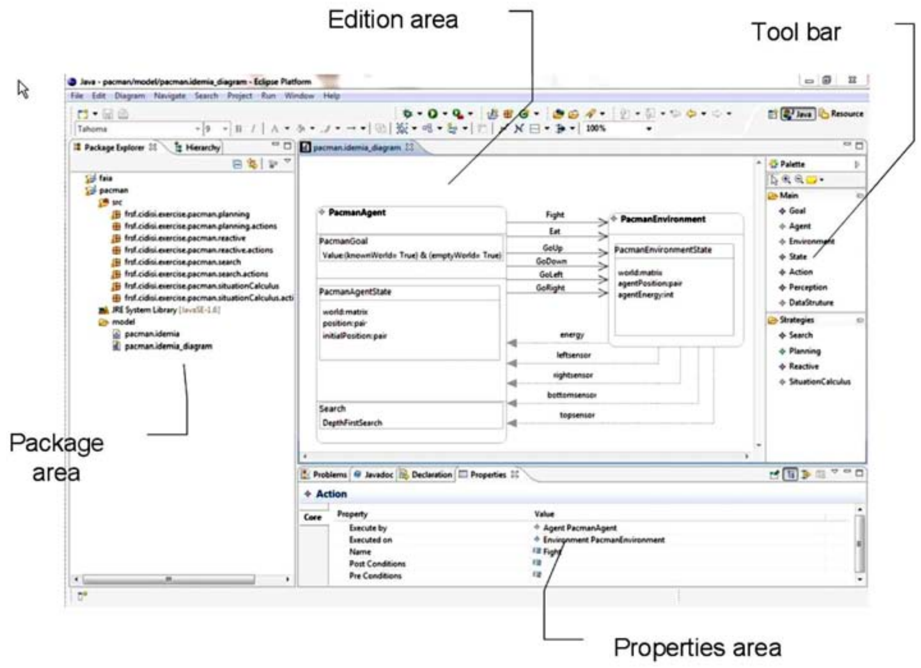

Based-object agent modeling Plug–in
Java code generation Plug –in
GMF
JET
FAIA
EMF
ECLIPSE PLATFORM
IDEM-IA

Figure 3. Architecture of IDEM-IA.

Finally, the upper layer of the architecture contains the main logic of IDEM-IA which has the plug-ins developed to support the requirements that motivated the development of this tool. This layer is composed of the plug-in for modeling software agents and the plug-in for generating Java code. The former defines the graphical editor for the conceptual development of software agents. This editor provides the graphical components of the concepts defined in the metamodel of the ACML language. The latter is used to perform the transformation of model to code using the services provided by the JET framework. The final result of these transformations is a set of classes extending the classes provided by FAIA.

With IDEM-IA, students can design agents with a familiar vocabulary representing the concepts taught in class. Agent models created with this tool not only allow the student to easily communicate their design solution to the professor, but it is also a bridge to fill the gap between the theoretical and the practical classes of AI. It is important to mention that, at the moment, the code generated by IDEM-IA is not the definitive agent solution since the student has to complete the generated code with specific implementation details in order to get a fully executable agent. Despite this, IDEM-IA is an important tool for students so that they can reduce the implementation time of their projects.

Figure 4 shows a project in IDEM-IA where there is an agent representing the classical Pac-Man. The *package area* shows the project, packages, files and folders related to a java project. At the bottom of this figure, we can observe the *Model* package with two files: *pacman.idemia* and *pacman.idemia_diagram*. The *Tool bar* area shows the available tools to edit an agent diagram. These elements correspond to the ACML language elements such as *Goal*, *Agent*, *Environment*, among others. The *Edition* area is the panel where the student develops its agent model. Finally, the *Properties* area allows students to modify the properties of the agent model such as the name of an agent, pre and post conditions of an action, etc. To generate the model, the elements in tool bar must be drag and drop in the edition area.



Property	Value
Execute by	Agent PacmanAgent
Executed on	Environment PacmanEnvironment
Name	Fight
Post Conditions	
Pre Conditions	

Figure 4. IDEM-IA.

Once the agent model is defined, it is possible to generate the corresponding Java code. The *pacman.idemia_diagram* files comprise the model, so doing right click on this file, the menu shown in Figure 5(a) is displayed. From this menu, we can select the option *Generate FAIA code* in order to produce the java classes related with the concepts belonging to the conceptual model. The package corresponds with the open project, in this case *Pacman package*, contains the generated classes. Figure 5(b) shows the *Pacman* package and the classes. Looking inside this figure, we can recognize classes related with the elements in the conceptual model shown in Figure 4. For instance, we can recognize among others that *Pacman-Agent* in the conceptual model corresponds to *PacmanAgent.java* class, *PacmanEnvironment* corresponds to *PacmanEnvironment.java* and *fight* action in conceptual model corresponds to *fight.java* class.

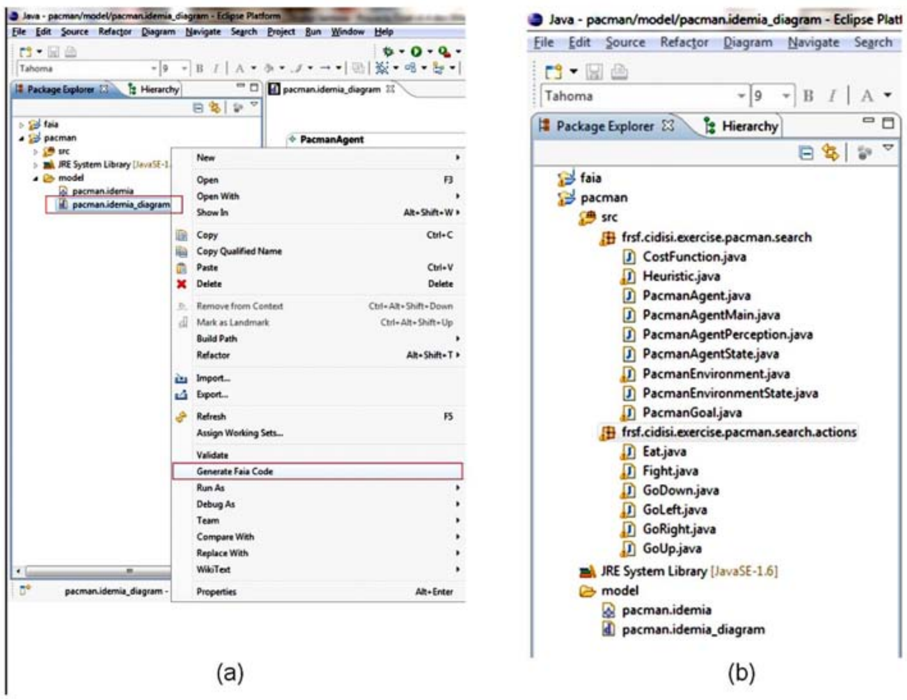


Figure 5. Code generation.

3.2. Framework for developing intelligent agents (FAIA)

In order to support the generation of the agent's executable code, we defined a set of rules which generate Java[4] code based on the FAIA framework[5].

FAIA is a framework we developed in previous work [16] with the aim of helping students in the implementation phase of software agents. It was developed as a practical framework that encloses the most important concepts of intelligent agents according to the traditional AI book of Russell and Norvig [1]. It provides a partial design of an agent which helps to avoid pitfalls in the development of students projects, and at the same time, it is kind of a guide to students that directs the agent development in the correct way.

FAIA makes it easier going from theory to implementation in a programming language. It has been used in the AI course at UTN-FRSF since 2008 with excellent results. Its use reduced mistakes and necessary re-work, and at the same time, helped students to: design the software solution, select the strategy to be used, understand the modular decomposition of an agent, understand the interaction between agents and environment, finish the practical work on due time and in full, and finally, show the results in a suitable graphical way. FAIA also helped professors to: evaluate the results, base the evaluation on homogeneous design-solutions, evaluate the students with more complex problems, and correct the practical work quickly.

4. Case Study

In 2011, we proposed students a project taken from a real world case study of the domain of Collaborative Business Processes [17]. Basically, students had to develop an agent to carry out the execution of a collaborative business process where two organizations must agree on a collaborative demand forecast. The scenario consists in two collaborating organizations.

[4]http://www.java.com
[5]http://code.google.com/p/faia

The organization "TK Computers", who plays the role of supplier, and "Computer's Market", who plays the role of customer.

The process management of collaborative demand forecast starts with the client, who asks the supplier for a demand forecast of a product. In order to provide a response, an employee of the Sales department uses an internal forecasting system to generate the response. Then the supplier processes the response and has two options. The supplier may agree and commit to carry out the demand forecast, or it may refuse the demand forecast and the process ends with a failure. If the supplier accepts the response, then the client must send independently the following information: the sale forecast generated by the client for each of the five point of sales for the considered product, and a plan for programmed events. With this information, the supplier generates a demand forecast, which is sent to the client. The supplier must respond to the client within five days starting from the demand forecast request. If the supplier does not respond in time, then an exception must be raised, where the client must send a cancel, and then the process ends.

When the customer updates the forecast and sends it to the supplier, a new negotiation cycle starts, where the supplier must send a new proposal and the customer can accept it or update it again. It is assumed that there are at most five negotiation cycles. Hence, if after five cycles there is no agreement, then the supplier must send a message pointing out the issues.

Figure 6 shows a student solution for this practical work. In this case, students developed a search based agent to represent the customer organization. The environment was interpreted as the supplier organization, from which to get perception and execute actions.

The agent, *CustomerAgent* has a *Customer-Goal*, *CustomerAgentState* and *Search* strategy associated. The *CustomerGoal* has a variable call *state* which is initialized with the string *Accept-Proposal*. The *CustomerAgentState* has five variables: *currentState*, *elapsedTime*, *forcastRequestDeliveryTime*, *negotiationCycle* and *informDeliveryTime*. The search strategy selected is *UniformCostSearch*.

The environment *Supplier* has a *SupplierState* defined with 6 variables: *states*, *negotiationCycle*, *lastActivityTime*, *informDeliveryTime*, *currentState* and *forecastRequestDeliveryTime*.

We can also see the actions that have been defined such as: *GenerateDemandForectasRequest*, *SendForecastRequestMessage*, *Failure*, *GenerateSaleForecastPOS*, among others.

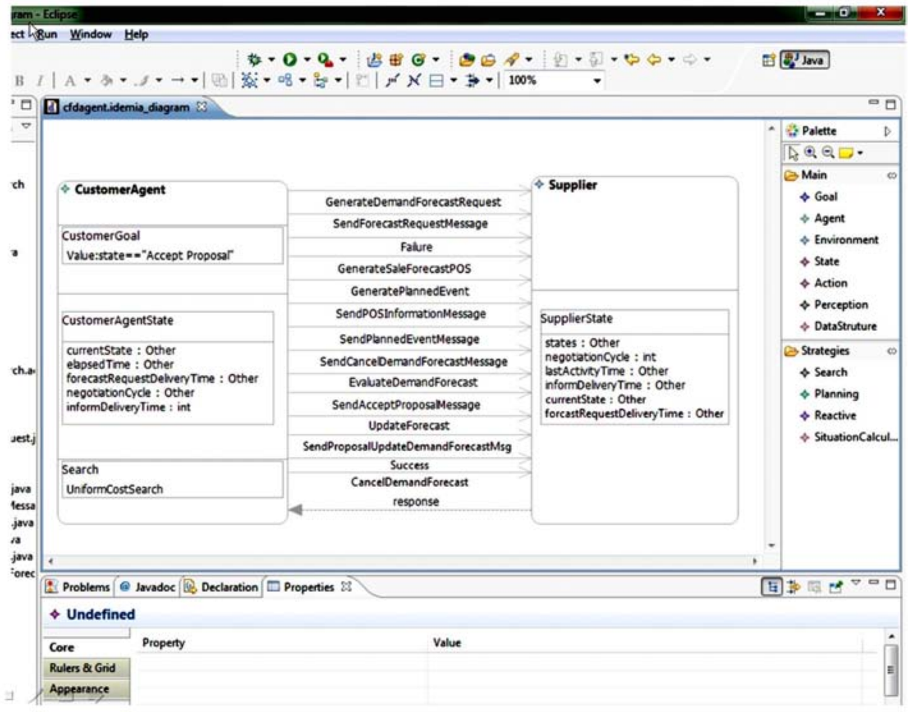
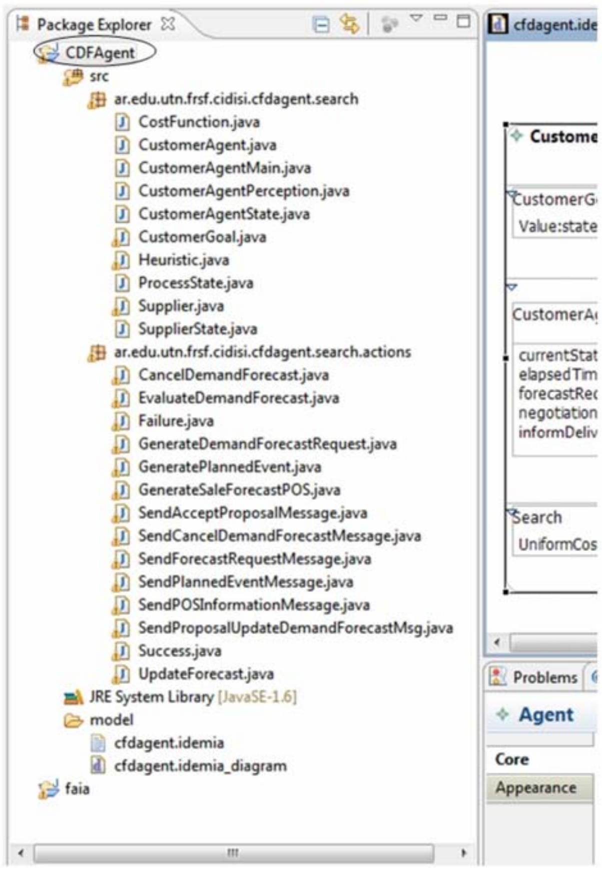


Figure 6. Case study: a collaborative demand forecast scenario.

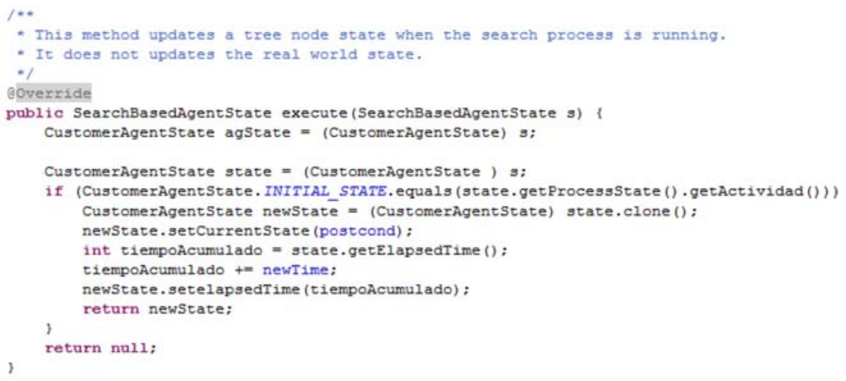
It has been defined one perception *response* corresponding to the supplier response.

Once the conceptual model is finished, the code generation function must be executed and the classes shown in Figure 7 are generated. But they are not complete in the sense that they have some methods without code. For instance, students must complete the *execute* method of actions. Figure 8 shows the *execute* method of the *GenerateDemandForecastRequest* action.



```
Package Explorer
CDFAgent
  src
    ar.edu.utn.frsf.cidisi.cfdagent.search
      CostFunction.java
      CustomerAgent.java
      CustomerAgentMain.java
      CustomerAgentPerception.java
      CustomerAgentState.java
      CustomerGoal.java
      Heuristic.java
      ProcessState.java
      Supplier.java
      SupplierState.java
    ar.edu.utn.frsf.cidisi.cfdagent.search.actions
      CancelDemandForecast.java
      EvaluateDemandForecast.java
      Failure.java
      GenerateDemandForecastRequest.java
      GeneratePlannedEvent.java
      GenerateSaleForecastPOS.java
      SendAcceptProposalMessage.java
      SendCancelDemandForecastMessage.java
      SendForecastRequestMessage.java
      SendPlannedEventMessage.java
      SendPOSInformationMessage.java
      SendProposalUpdateDemandForecastMsg.java
      Success.java
      UpdateForecast.java
  JRE System Library [JavaSE-1.6]
  model
    cfdagent.idemia
    cfdagent.idemia_diagram
faia
```

Figure 7. Case study: code generation.



```
/**
 * This method updates a tree node state when the search process is running.
 * It does not updates the real world state.
 */
@Override
public SearchBasedAgentState execute(SearchBasedAgentState s) {
    CustomerAgentState agState = (CustomerAgentState) s;

    CustomerAgentState state = (CustomerAgentState ) s;
    if (CustomerAgentState.INITIAL_STATE.equals(state.getProcessState().getActividad()))
        CustomerAgentState newState = (CustomerAgentState) state.clone();
        newState.setCurrentState(postcond);
        int tiempoAcumulado = state.getElapsedTime();
        tiempoAcumulado += newTime;
        newState.setelapsedTime(tiempoAcumulado);
        return newState;
    }
    return null;
}
```

Figure 8. Example of the *execute* method of the G*enerateDemand-ForecastRequest* action.

5. Evaluation and Students Perception

The condition to promote the previously mentioned AI course are:

- Attending 80% of classes;
- Doing a practical work (AI agent design + implementation) achieving a score higher than (or equal to) 80 points (over 100);
- Taking a midterm exam, achieving a score higher than (or equal to) 80 points (over 100).

If students do not meet the minimum requirements but they achieve a score higher than (or equal to) 60 points (over 100), then they reach the regular condition. In this case, students have to take a final exam in order to promote the subject. In any other cases, students must retake the course (free condition). The goal is, of course, to promote the course in the shortest instance. However, one of the problems that professors have detected in achieving the goal is that students cannot do the proposed practical work in full and on time. We have asked students and have discovered that this problem was mainly caused by the long time needed to learn the use of the framework, added to the need to learn new concepts of AI and to implement the answer simultaneously in a short period of time (a term).

Facing this situation, we began working in the design of a computational tool that would allow students to design a solution to an AI problem by using AI concepts, while the design becomes a code solution, in an easy way. During the academic year 2010, IDEM-IA was being developed by professors of the Artificial Intelligence subject. Students had no opportunity to use it that year and they solved that term practical work (necessary for promoting the course) without it. During 2011 and the first quarter of 2012, IDEM-IA was already incorporated into the study material of the AI course. This section reports the results achieved. The practical work in 2011 has been described in case study section. In 2012, the practical work was the development of a chatbot which simulates an AI student. The goal is that the chatbot must be capable of chatting with a student and answer questions about AI class as, for example: when the midterm exam is, who the teacher is, when the dead-line for practical work is, among others. The students not only have to develop the agent but they also have to define the vocabulary and the questions that the agent can understand.

Last year, the best answer of the practical work was published in a student congress (40 JAIIO - EST 2011)[6] and this year the best answer has been accepted as well in the same congress that will be held in La Plata - Argentina on August (41 JAIIO EST 2012)[7].

In both 2011 and 2012, all students used the tool with good results. They manifested that IDEM-IA is an intuitive and easy to use tool, and that helped them to understand and use the FAIA framework. The generated code has guided students to a more deeply understanding of the relations and architecture of the implemented solution, reducing the time needed to learn the framework.

Table 1. Results of IDEM-IA use

Student situation	2010	2011	2012
Attending students	42	41	45
Promoted	20%	61%	85%
Regular	76%	37%	12%
Free	4%	2%	3 %

Table 1 shows in numbers the improvement in promoted students that have attended Artificial Intelligent courses in 2010, 2011 and 2012. We can see that in 2010 the attendees students were 42, 20% of which was promoted the AI course, the 76% was regular and 4% was free. In 2011, on a total of 41 students, 61% was promoted, while only 37% was regular and 2% was free. As regards 2012, on the 45 students, 85% was promoted, 12% was regular and 3% was free.

[6]http://www.40jaiio.org.ar/
[7]http://www.41jaiio.org.ar/est

6. Conclusions and Future Work

We have presented an educational tool based on a model-driven methodology in order to teach artificial intelligence in an engineering discipline. We have described the language used by the tool to define an agent model. We have emphasized the suitability in the use of this type of tool that allows students reduce the gap between theory and implementation solution. Results on the 2010, 2011 and 2012 were shown, highlighting the achieved improvement. As future work, we are working to adding new functionalities to provide the development of different type of agents and make this tool flexible enough to use with different programming languages and tools.

References

[1] S. Russell and P. Norvig, Artificial Intelligence - A Modern Approach, 3rd ed., Pearson Education, 2010.

[2] D. Kumar and L. Meeden, A robot laboratory for teaching artificial intelligence, SIGCSE, 1998, pp. 341-344.

[3] J. Bennedsen, M. E. Caspersen and M. Kölling, eds., Model-driven programming, Lecture Notes in Computer Science, Springer, Vol. 4821, 2008.

[4] M. Baker, The roles of models in artificial intelligence and education research: a prospective view, Inter. J. Artificial Intelligence in Education 11 (2000), 122-143.

[5] J. Siegel, Developing in OMG’s model-driven architecture, Technical Report, OMG, 2001.

[6] J. Koehler, R. Hauser, J. Kuster, K. Ryndina, J. Vanhatalo and M. Wahler, The role of visual modeling and model transformations in business-driven development, Electronic Notes in Theoretical Computer Science, Vol. 211, 2008, pp. 5-15.

[7] Y. Ni and Y. Fan, Model transformation and formal verification for semantic web services composition, Advances in Engineering Software 41(6) (2010), 879-885.

[8] P. Bocciarelli and A. Dambrogio, A model-driven method for describing and predicting the reliability of composite services, Software and Systems Modeling 10(2) (2010), 265-280.

[9] H. Zha, W. van der Aalst, J. Wang, L. Wen and J. Sun, Verifying workflow processes: a transformation-based approach, Software Systems Modelling 10(2) (2011), 253-264.

[10] A. Hamou-Lhadj, A. Gherbi and J. Nandigam, The impact of the model-driven approach to software engineering on software engineering education, Sixth International Conference on Information Technology: New Generation, 2009, pp. 719-724.

[11] J. Bezivin, R. B. France, M. Gogolla, O. Haugen, G. Taentzer and D. Varro, Teaching modeling: why, when, what? MoDELS Workshop, 2009, pp. 55-62.

[12] Eclipse platform. [Online]. Available: http://www.eclipse.org

[13] Eclipse modeling framework. [Online]. Available: http://www.eclipse.org/modeling /emf

[14] Graphical modeling framework. [Online]. Available: http://www.eclipse.org/modeling /gmf

[15] Java emitter templates. [Online]. Available: http://www.eclipse.org/modeling/m2t/?project=jet#jet

[16] J. Roa, M. Pividori, M. Gutierrez and G. Stegmayer, How to develop intelligent agents in an easy way with FAIA, Ed. EGI Global, 2010.

[17] P. Villareal, I. Lasarte, J. Roa and O. Chiotti, A modeling approach for collaborative business processes, Business Process Management Workshops 43 (2010), 318-329.