ISSN 0103-9741

Monografias em Ciência da Computação
nº 11/09

Towards a Model Driven Approach for Engineering Self-Organizing Multi-Agent Systems

Maíra Athanázio de Cerqueira Gatti
Ugo Braga Sangiorgi
Carlos José Pereira de Lucena

Departamento de Informática

Monografias em Ciência da Computação, No. 11/09 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena March, 2009

Towards a Model Driven Approach for Engineering Self-Organizing Multi-Agent Systems

Maíra Athanázio de Cerqueira Gatti, Ugo Braga Sangiorgi and Carlos José Pereira de Lucena

Laboratório de Engenharia de Software – LES
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brasil

{mgatti, usangiorgi, lucena}@inf.puc-rio.br

Abstract. This paper introduces a tool for modeling self-¬organizing multi¬-agent systems based on a design approach that contains models that allow the design of more than message passing but the emergence of highly decentralized and dynamic distributed systems. It is crucial for self-organizing systems to model an environment where agents can interact indirectly through intentional events, for example by leaving objects in an environment for other agents to see. It is more scalable and convenient for the application developer. Our tool allows the visual construction and validation of the proposed models, representing the first step for code generation.

Keywords: Model Driven Development, Multi-Agent Systems, Self-organization, Coordination.

Resumo. Este artigo apresenta uma ferramenta de modelagem de sistemas multi-agentes auto-organizáveis baseada em uma abordagem de design que possui modelos que permitem o projeto de propriedades emergentes de sistemas altamente dinâmicos e descentralizados. A modelagem do ambiente onde os agentes interagem de forma indireta através de eventos intencionais, como por exemplo deixando objetos no ambiente para que outros agentes percebam, é crucial para sistemas auto-organizáveis. Além de ser mais escalável e conveniente para o desenvolvedor da aplicação. A ferramenta aqui proposta permite a construção visual e validação dos modelos propostos, representando um primeiro passo para geração de código.

Palavras-chave: Desenvolvimento Orientado a Modelos, Sistemas Multiagentes, Auto-Organização, Coordenação.

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

Sumário

1 Introduction 1

2 The Model Driven Approach 1

2.1 The Static Meta-Model Editor 2

2.2 The Dynamic Meta-Model Editor 3

3 The Self-Organizing Framework Instantiation and Source Code Generation 6

4 Related Work 7

5 Discussion and Future work 8

References 9

1 Introduction

Self-organizing systems tend to be decentralized and bottom-up driven [9] because self-organization is a dynamic and adaptive process where components of a system acquire and maintain information about their environment and neighbors without external control. This makes it more difficult to compare such a system's design and maintenance to non-self-organizing systems.

On the other hand, Agent-Oriented Software Engineering (AOSE) can provide methods and technologies that help in the building of self-organizing systems. In [14-15], a design approach is proposed to allow the modeling of more than message passing but the emergence of highly decentralized and dynamic distributed systems.

A static and a dynamic model was proposed; the former reuses the UML class model, and the latter the UML state diagrams. The problem this work focuses on practical aspects of modeling the agent systems. There is a need for a specific editor for the language, since drawing the models by hand can be a really time-consuming task, even for simple models. Furthermore, the artifact produced in this manner only contains drawings; there is no semantic on the models, hence it is not possible to automatically validate them.

Therefore, to have a specific editor to design the models would contribute not only to faster modeling of static and dynamic models, but also would lead us to a better understanding of emergent properties of the self-organizing systems onto the proposed language. In addition, to have well defined models leads the way for code generation in the future.

This paper is organized as follows: the next Section presents the chosen technologies that underpin the description of our models and their editors. Section 3 describes the framework instantiation and how code generation can be done for a self-organizing framework [16]. Section 4 presents the related works, and finally we present the conclusions and future development.

2 The Model Driven Approach

The model driven approach consists of a meta-model to structure entities, a representation model based on coordination statecharts [15] which adapts UML 2 diagrams [19],[11],[5], an editor to support the modeling, and a Java-based source code generation with the automatic instantiation of a self-organizing simulation framework.

We had developed a group of Eclipse plug-ins for the editor, mainly using GMF (Graphical Modeling Framework)[1] , which allows a rapid development of graphical editors based on GEF (Graphical Editor Framework)[2] for domain models specified with EMF (Eclipse Modeling Framework)[3]. All these frameworks combined allow a user to visually develop a domain model represented in a compliant XMI.

Both the dynamic and static models were written using EMF, extending the Eclipse UML2 framework model, just like the meta-models extend UML's model. According to

1 http://www.eclipse.org/modeling/gmf

2 http://www.eclipse.org/gef

3 http://www.eclipse.org/modeling/emf

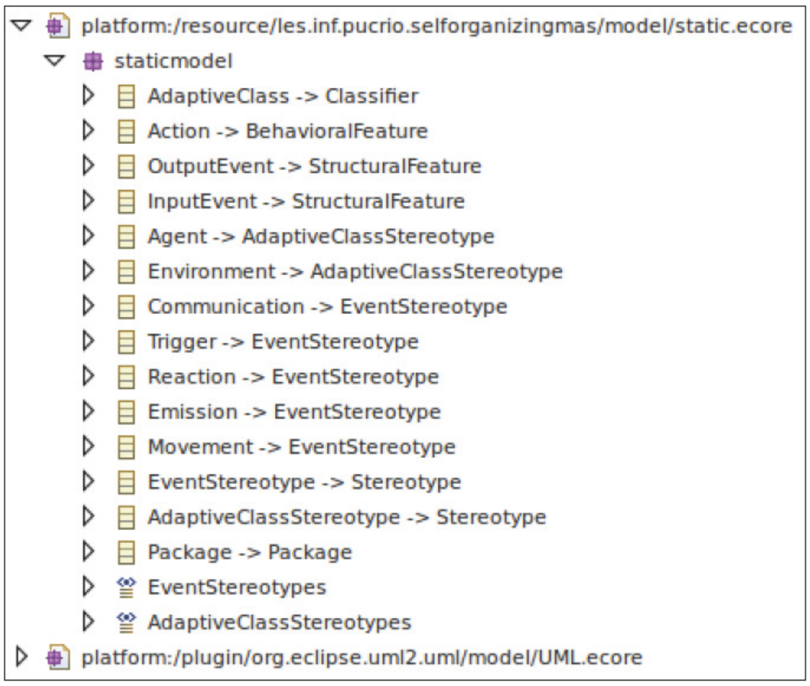
[4], extensions to UML Tools meta-models may be done using a lightweight, a middleweight or a heavyweight approach. We had chosen to use the middleweight, since the lightweight uses profiles and stereotypes, and therefore it was impossible to redefine structure or behavior, and using the heavyweight one involves reuse by "copy and merge" instead of reuse by specializing types from the meta-model, which could aggregate more complexity to the work and lose interoperability with other UML tools.

Thus, we used a middleweight solution since we reused and adapted the UML meta-model, creating our own state editor to support the coordinated statecharts. We had built the editor trees for the static and dynamic models in order to allow a first validation of the tool, hence the visual editor can be developed. The models are briefly described while the static and dynamic meta-model editors are presented.

2.1 The Static Meta-Model Editor

The meta-class Adaptive must be stereotyped either as agent or environment. The Adaptive class can be either a proactive or reactive agent with sensors and effectors. An agent can execute several actions regarding its goals or perceptions. As well, the environment has the same features.

There are two structural features, the *input* and *output event*, that define the events that the agent or environment perceives (input event) and that is a condition for activating an action, and the events that they generate (output event) after executing the action. They vary according to the stereotypes and can be stereotyped according to the event type.



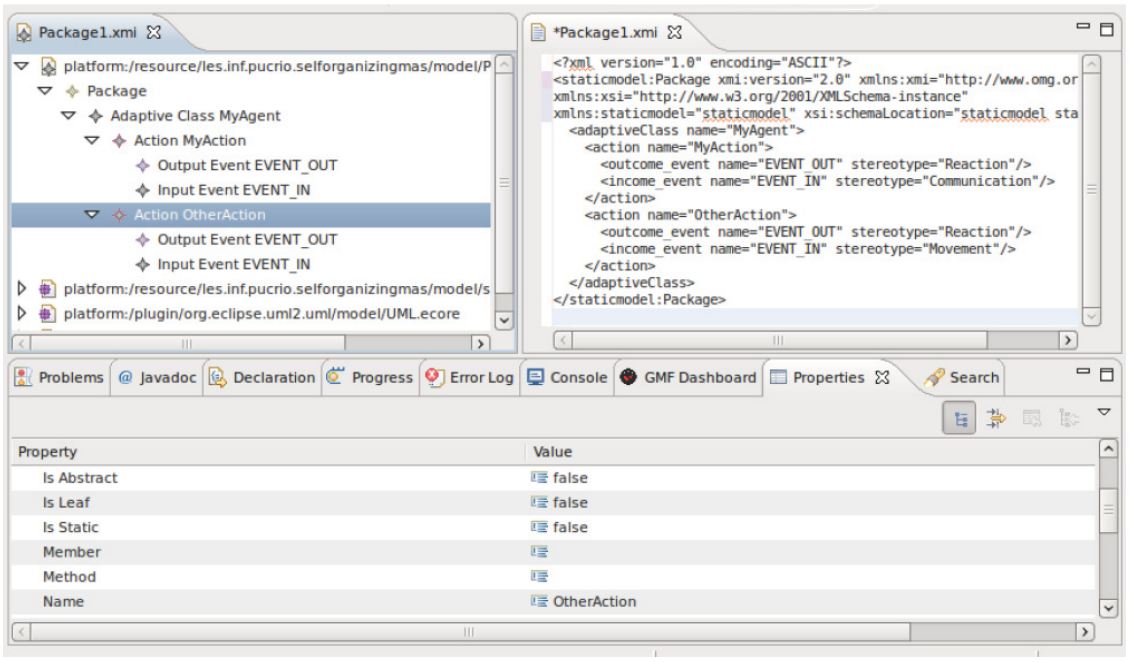
```
platform:/resource/les.inf.pucrio.selforganizingmas/model/static.ecore
  staticmodel
    AdaptiveClass -> Classifier
    Action -> BehavioralFeature
    OutputEvent -> StructuralFeature
    InputEvent -> StructuralFeature
    Agent -> AdaptiveClassStereotype
    Environment -> AdaptiveClassStereotype
    Communication -> EventStereotype
    Trigger -> EventStereotype
    Reaction -> EventStereotype
    Emission -> EventStereotype
    Movement -> EventStereotype
    EventStereotype -> Stereotype
    AdaptiveClassStereotype -> Stereotype
    Package -> Package
    EventStereotypes
    AdaptiveClassStereotypes
platform:/plugin/org.eclipse.uml2.uml/model/UML.ecore
```

Figure 1. The static meta-model

The *action* behavior feature is executed during agent or environment execution without explicitly being called by other objects or agents. Agents interact with one another and the environment, sending and receiving messages or sending and receiving events through propagations in the environment.

Every action is described specifying preconditions and effects. Like the Operation meta-class, the Action meta-class is associated with the Constraints meta-class in order to define the pre and post conditions and the in and output events. Constraints are conditions expressed in natural language text or in a machine legible language in order to declare an entity's semantics [19].

The static meta-model developed is illustrated in Figure 1. Notice the extensions to the EMF meta-model like the *StructuralFeatures*, *BehavioralFeatures* and *Stereotypes*. From this meta-model, an editor tree was generated (Figure 2). It is possible to add specific elements to this editor (beyond the UML meta-model elements) in order to compose an appropriate model to the proposed specification.



```
platform:/resource/les.inf.pucrio.selforganizingmas/model/P
  Package
    Adaptive Class MyAgent
      Action MyAction
        Output Event EVENT_OUT
        Input Event EVENT_IN
      Action OtherAction
        Output Event EVENT_OUT
        Input Event EVENT_IN
platform:/resource/les.inf.pucrio.selforganizingmas/model/s
platform:/plugin/org.eclipse.uml2.uml/model/UML.ecore
```

```
<?xml version="1.0" encoding="ASCII"?>
<staticmodel:Package xmi:version="2.0" xmlns:xmi="http://www.omg.or
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:staticmodel="staticmodel" xsi:schemaLocation="staticmodel sta
  <adaptiveClass name="MyAgent">
    <action name="MyAction">
      <outcome_event name="EVENT_OUT" stereotype="Reaction"/>
      <income_event name="EVENT_IN" stereotype="Communication"/>
    </action>
    <action name="OtherAction">
      <outcome_event name="EVENT_OUT" stereotype="Reaction"/>
      <income_event name="EVENT_IN" stereotype="Movement"/>
    </action>
  </adaptiveClass>
</staticmodel:Package>
```

Property	Value
Is Abstract	false
Is Leaf	false
Is Static	false
Member	
Method	
Name	OtherAction

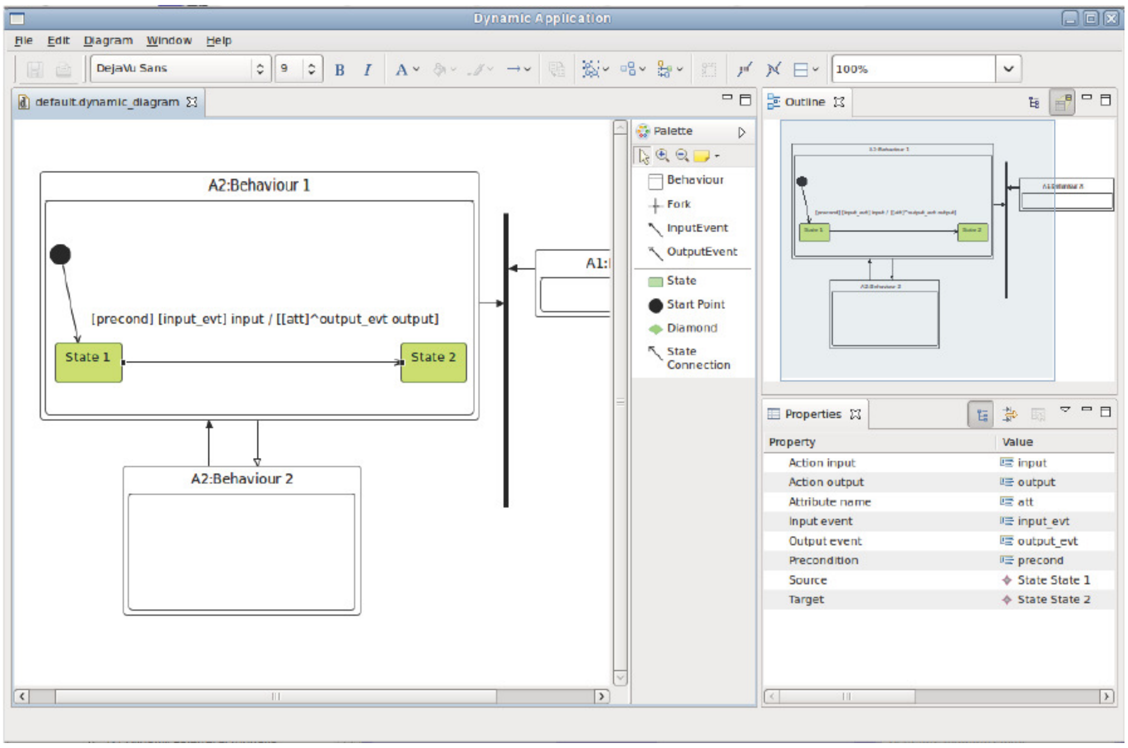
Figure 2. The static meta-model editor.

2.2 The Dynamic Meta-Model Editor

Coordination is defined as the management of the communication between agents – coordination defines how agents interact. In self-organizing systems this interaction occurs through the environment (e.g. gradient fields, pheromones) [**Error! Reference source not found.**],[**Error! Reference source not found.**]. Furthermore a coordination design approach allows the design of more than message passing but the emergence of highly decentralized and dynamic distributed systems. It is crucial for self-organizing systems to model an environment where agents can interact indirectly through intentional events, for example, by leaving objects in an environment for other agents to see, and it is more scalable and convenient for the application developer.
Coordinated statecharts is based on the UML State Machine diagram. It combines statecharts [18]) with action and communication of behaviors. A State Machine Diagram describes discrete behavior modeled through finite state-transition systems. More specifically, coordinated statecharts reuses the UML 2 behavioral state machine. Each agent and environment behavior is designed using behavioral state machine diagrams. Each behavioral state machine diagram can communicate with all the other diagrams through a communication channel and the desired emergent property may appear as a result of that communications/coordination. Moreover, behaviors are composed of actions. Actions are executed through input events and pre-conditions and raise output events.

Coordinated statecharts compose behaviors in parallel. With coordinated statecharts, a behavior is a particular instance of the agent or environment in a scenario that represents a typical path through the state space within a single state machine, i.e., an ordered sequence of state transitions triggered by events and accompanied by actions. A notable problem with a traditional state machine is that it allows only a single behavior within itself at a time, thus executing concurrent behaviors (for each agent/environment) only in a sequential manner on a behavior-by-behavior basis, rather than on an event-by-event basis. The result can be poor real-time performance and underutilization of system resources such as the CPU and network bandwidth.
The rationale behind coordinated statecharts is that it can interleave the execution of concurrent behaviors by switching among behaviors on an event-by-event basis. As well, the self-organizing mechanism may reuse all or part of local behaviors. Thus, the new dynamic model representation must be able to encapsulate behaviors in such a way that they can be reused. Therefore, the semantics of coordinated statecharts is defined by the semantics of state diagrams and descriptions in natural languages.

Coordinated statecharts allows the design of information-oriented indirect communication. It allows the modularity and reuse of local behaviors. And it allows the design and reuse of self-organizing architectural patterns. Moreover coordinated statecharts allows the design of feedback loops [**Error! Reference source not found.**],[**Error! Reference source not found.**].



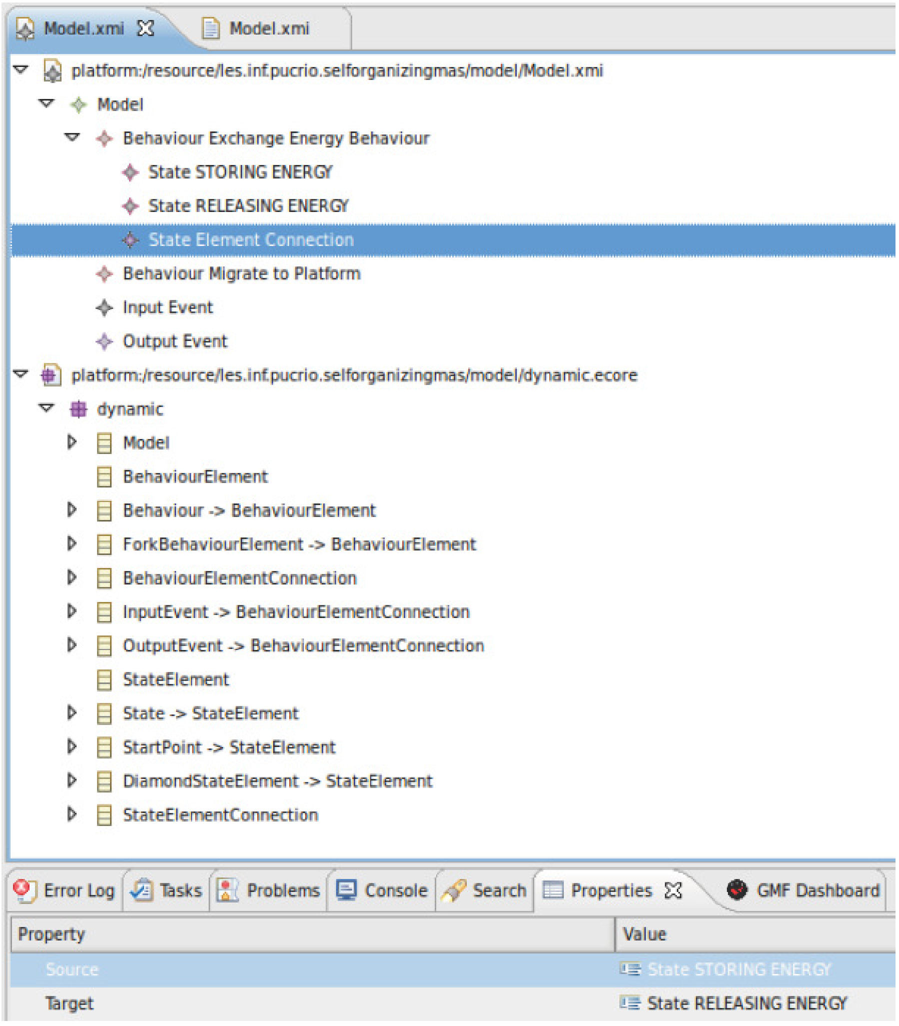
Property	Value
Action input	input
Action output	output
Attribute name	att
Input event	input_evt
Output event	output_evt
Precondition	precond
Source	State State 1
Target	State State 2

Figure 3. The dynamic meta-model editor

The dynamic model editor tree is identical to a state machine editor, with the addition of the facility of transitions edition. As the transition syntax might change in the future and is very long, we showed the visible attributes in a different way (right inferior part of the Figure 3). Therefore, the visualization is flexible enough, in the future, to allow the removal or shortening of very extensive attributes.

In the same way, the straight edition of the transition is allowed, and the attributes are filled in the right inferior part automatically — of course, respecting the syntax. Therefore, the editor handles an XML file that contains an instance of the meta-model:

```
<?xml version="1.0" encoding="UTF8"?>
<dynamicmodel:Model xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xmlns:dynamicmodel="dynamicmodel">
     <elements xsi:type="dynamicmodel:Behaviour" name="Behaviour 1"
      agent_instance="A2">
          <elements xsi:type="dynamicmodel:State" name="State 1"/>
          <elements xsi:type="dynamicmodel:State" name="State 2"/>
          <elements xsi:type="dynamicmodel:StartPoint"/>
          <transition source="//@elements.0/@elements.0"
          target="//@elements.0/@elements.1" precondition="precond"
          input_event="input_evt" attribute_name="att" output_event="output_evt"
          action_output="output" action_input="input"/>
          <transition source="//@elements.0/@elements.2"
          target="//@elements.0/@elements.0"/>
     </elements>
```



```
Model.xmi    Model.xmi
platform:/resource/les.inf.pucrio.selforganizingmas/model/Model.xmi
  Model
    Behaviour Exchange Energy Behaviour
      State STORING ENERGY
      State RELEASING ENERGY
      State Element Connection
    Behaviour Migrate to Platform
    Input Event
    Output Event
platform:/resource/les.inf.pucrio.selforganizingmas/model/dynamic.ecore
  dynamic
    Model
    BehaviourElement
    Behaviour -> BehaviourElement
    ForkBehaviourElement -> BehaviourElement
    BehaviourElementConnection
    InputEvent -> BehaviourElementConnection
    OutputEvent -> BehaviourElementConnection
    StateElement
    State -> StateElement
    StartPoint -> StateElement
    DiamondStateElement -> StateElement
    StateElementConnection
```

Property	Value
Source	State STORING ENERGY
Target	State RELEASING ENERGY

Figure 4. The dynamic meta-model

Also, an editor tree was generated for the dynamic meta-model in a way that what is seen in the editor tree is the same model in the coordinated statecharts (dynamic) editor. In Figure 4 the model illustrates how the structure is displayed. For instance, it shows a model that contains a *Behavior* with three *States* and two events that connect to other *Behavior*.

3 The Self-Organizing Framework Instantiation and Source Code Generation

In this section we show how the models can be realized into code through the instantiation of a multi-environment self-organizing simulation framework [**Error! Reference source not found.**]. It supports modeling and development of self-organizing and self-adaptive systems; it allows multi-environment simulation; it allows both 2D and 3D discrete and continuous visualization.

The framework provides higher abstractions and components to support the development of self-organizing systems with multiple environments. There is one interface for agents and one for environments. The framework provides two classes for each environment type to be developed (situated/ non-situated, discrete/ continuous, 2D/ 3D, and their combination) that implement those interfaces and they provide a set of reusable behaviors that can be used for any application to be instantiated.

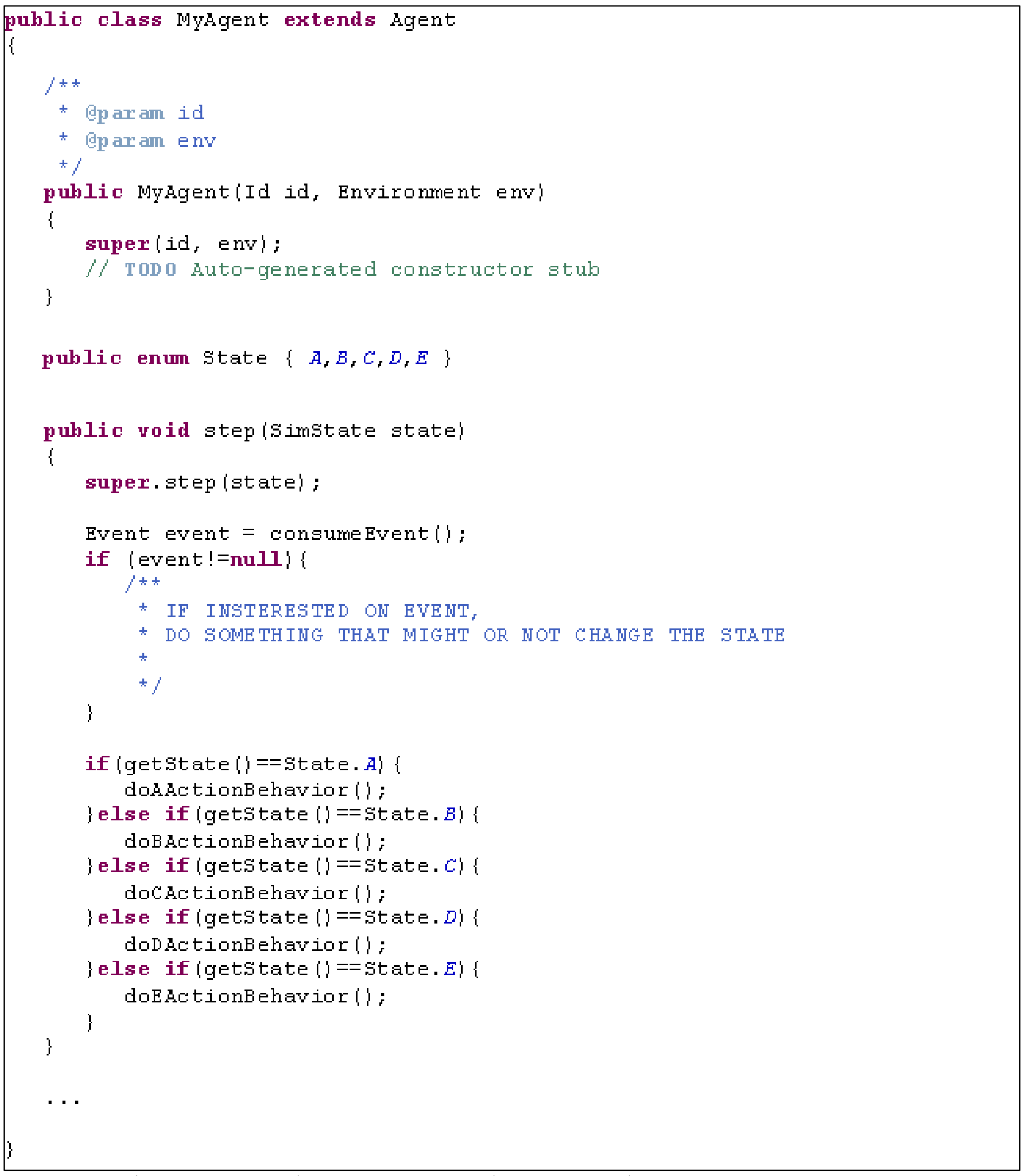
In a discrete event simulation system, an entity is allowed to behave from time to time. These slices of time are called steps. So, a basic entity will usually implement the step method where it will perform its activities. The agent or environment has a set of events, i.e., the information provided for the underline self-organization and implicit coordination, to handle on each time step of simulation. An agent can behave and execute actions over the environment where it resides or over itself.

For instance, considering a 2D environment, the framework provides the *Agent2D* and *Environment2D* classes for situated environment using a discrete 2D double point grid that is represented by the class *Grid2D*. This class handles the addition, removal and search of agents and events in a double point location. The environment uses the grid to realize the several strategies for self-organizing patterns, such as the atomic ones Replication, Death, Diffusion, and Aggregation [**Error! Reference source not found.**], or combined ones, such as Gradient Fields and Pheromone Path [**Error! Reference source not found.**], for instance.

Generally, for a non-situated environment, the models and coordinated statecharts can be realized into code following the abstract instantiation below:

- Agents must extend the abstract class *Agent*;
- Environment must extend the abstract class Environment;
- Simulator controller must instantiate the Environment;
- Transitions are fired per step according to preconditions and input events;
- States are defined as enumeration constants;
- Events must be typed using the Event.Type enum constant;
- Event-trigged actions are executed as a match of *Event* and *State*.

And the Figure 5 illustrates the Java code for an abstract instantiation. Imagine that an instantiated agent of the class *MyAgent* is in the state *A*. For each simulation time step it checks for new events in the environment or sent directly to it. This event might cause a state change or not. Suppose that the agent state did not change. When it executes the *doAActionBehavior()* action means that it is on the *A* state and is executing the action specified for this state. Then, it fires an event and changes its state to *B*. Then, in the next simulator time step, it will execute the *doBActionBehavior()*, unless other agent or even the environment propagates an event that changes its state to a different one before executing it, if it was specified.



```
public class MyAgent extends Agent
{

   /**
    * @param id
    * @param env
    */
   public MyAgent(Id id, Environment env)
   {
      super(id, env);
      // TODO Auto-generated constructor stub
   }

   public enum State { A,B,C,D,E }


   public void step(SimState state)
   {
      super.step(state);

      Event event = consumeEvent();
      if (event!=null){
         /**
          * IF INSTERESTED ON EVENT,
          * DO SOMETHING THAT MIGHT OR NOT CHANGE THE STATE
          *
          */
      }

      if(getState()==State.A){
         doAActionBehavior();
      }else if(getState()==State.B){
         doBActionBehavior();
      }else if(getState()==State.C){
         doCActionBehavior();
      }else if(getState()==State.D){
         doDActionBehavior();
      }else if(getState()==State.E){
         doEActionBehavior();
      }
   }

   ...

}
```

Figure 5. The Source Code generation and Framework instantiation.

4 Related Work

In this section we present an overview of existing methodologies for multi-agent systems supported by tools with code generation for platforms such as Jade [**Error! Reference source not found.**].

Tropos is a requirement-oriented methodology for MAS [3]. It consists in three phases: requirement analysis, project and code generation. The tool TAOM4e (Tool for Agent Oriented Modeling)4 is shipped as an Eclipse plug-in and is model-oriented [2]. During the project step, the goal models are transformed by Tropos2UML to UML activity diagrams and then transformed to Jade code by the UML2Jade tool. The PASSI (Process for Agent Societies Specification and Implementation) describes the creation of five steps and models: requirements, agent modeling, agent implementation, code

4 http://sra.itc.it/tools/taom4e

model and deployment [8]. The PASSI ToolKit4(PTK)[5] is a plug-in for Rational Rose with models and diagrams based on UML and provides code generation for Jade.

The Gaia methodology was proposed by [**Error! Reference source not found.**] and allows a high level specification of MAS. Huang et al. proposed many diagrams for that methodology, but nowadays there is no tool to support them. The WADE (Workflows and Agents Development Environment) extends Jade agents for workflow execution [**Error! Reference source not found.**]. The Wolf Eclipse plug-in supports the development of WADE applications, providing a graphical editor of diagrams and a code editor.

5 Discussion and Future work

It is expected from the editors to have an epistemic function in a way that the modeling of multi-agent systems itself leads to a better understanding and observation of coordination mechanisms in self-organizing systems. Also, the tool could be used to model architectural patterns of self-organization [**Error! Reference source not found.**] optimizing the project time and effort for solutions that make use of those patterns (Figure 6).

Furthermore, it is desired to have automatic communication between both static and dynamic models, although we do not yet have it. Thus, the next step is to build an interface between the models that updates them in both directions, each time the models change since the programmatic interpreter of both models is already built-in. It populates objects for each element on the model, as well as updating the diagram view, whenever the XML underneath changes. Thus, an interface can provide ways of delegating such functions for the already built layer.

The next step, after the construction of update mechanisms, is the generation of code based on the built models, which is eased by the Eclipse platform, since the models are compatible with EMF framework and there are available mechanisms of reflection and model transformation.

5 http://sourceforge.net/projects/ptk

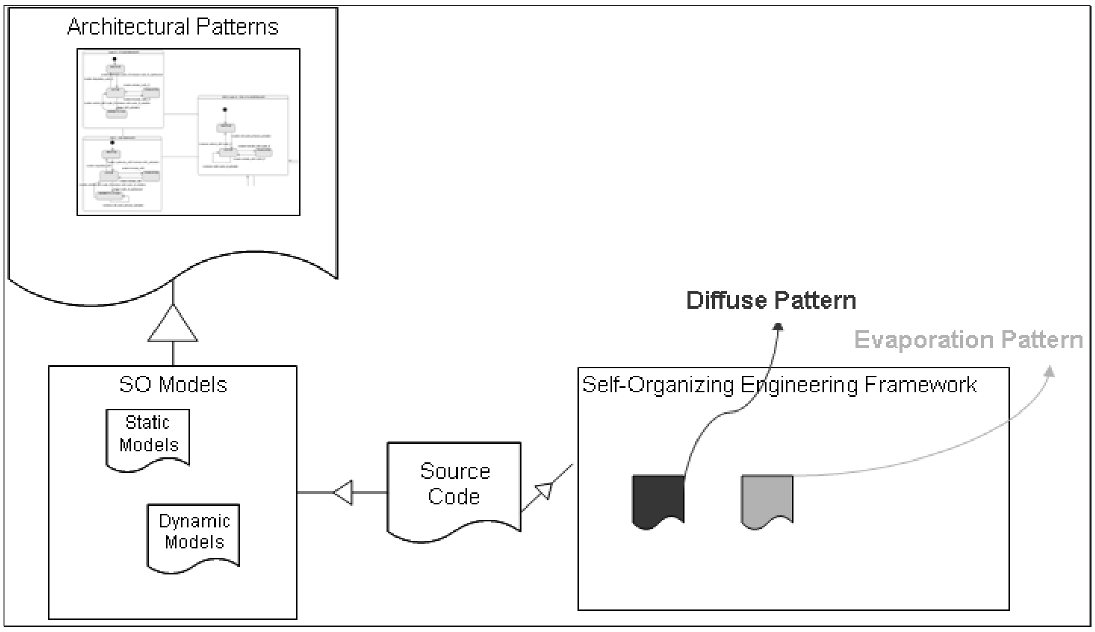


Figure 6. From the architectural design patterns and models to source code generation and framework instantiation.

ACKNOWLEDGMENT. This work was supported by MCT/CNPq through the "*Grandes Desafios da Computação no Brasil: 2006-2016*" (Main Computation Challenges in Brazil: 2006-2016) Project (Proc. CNPq 550865/2007-1).

References

[1] Bellifemine, F., Poggi, A., Rimassa, G.: Jade, A FIPA-compliant Agent Framework. Proceedings of PAAM'99, London, UK (1999)

[2] Bertolini, D., Novikau, A.., Susi, A.., and Perini, A.: TAOM4E: an Eclipse ready tool for agent-oriented modeling. Issue on the development process. Technical report, ITC-irst, 2006.

[3] Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J. and Perini A. Tropos: An agent-oriented software development methodology. Journal of Autonomous Agents and Multi-Agent Systems, 8(3):203–236, 2004.

[4] Bruck, J. Hussey, K http://www.eclipse.org/modeling/mdt/uml2/docs/articles/Customizing_UML2_Which_Technique_is_Right_For_You/article.html. Junho 2008.

[5] Booch, G., Rumbaugh, J., Jacobson, I.; Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Obj. Tech. Series). 2005.

[6] Camazine,S., Deneubourg,, J.-L. Franks, N. R., Sneyd, J., Theraula, G., Bonabeau, E.. Self-Organization in Biological Systems. Princeton University Press, 2003.

[7] Caire, G., Gotta, D. and Banzi, M.: Wade: a software platform to develop mission critical applications exploiting agents and workflows. In Proceedings of the 7th inter-

national joint conference on Autonomous agents and multiagent systems (AAMAS 2008), pages 29–36, 2008.

[8] Cossentino M. e Potts C. A CASE tool supported methodology for the design of multi-agent systems. In Proceedings of The 2002 International Conference on Software Engineering Research and Practice (SERP'02), 2002.

[9] Di Marzo Serugendo, G., Fitzgerald, J. S., Romanovsky, A., Guelfi, N. Generic Framework for the Engineering of Self-Adaptive and Self-Organising Systems. CS-TR-1018, 2007.

[10] De Wolf, T.; Analysing and engineering self-organising emergent applications, Ph.D. Thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, May, 2007, 183.

[11] Fowler, M.; UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition) (The Addison-Wesley Object Technology Series), 2004.

[12] FIPA: Foundation for intelligent physical agents (http://www.fipa.org/)

[13] Gardelli, L., Viroli, M., Omicini; A.; Design Patterns for Self-Organizing Multi-agent Systems. 2nd Int. Workshop on Eng. Emergence in Decentralised Autonomic Systems (EEDAS 2007). To be held at the 4th IEEE Int. Conf. on Autonomic Computing (ICAC 2007). June 11th, 2007, Jacksonville, Florida, USA.

[14] Gatti, M.A. de C., Lucena, C.J.P.; "A Bio-inspired Representation Model for Engineering Self-Organizing Emergent Systems," XXII Simpósio Brasileiro de Engenharia de Software (SBES), Campinas, SP, Brasil, 2008.

[15] Gatti, M.A. de C., Lucena, C.J.P.: Engineering Self-Organizing Multiagent Systems based on a Bio-inspired Representation Model and Coordinated Statecharts. Submitted to a Special Issue Track in the Information Sciences Journal, 25 pgs., 2009.

[16] Gatti, M.A. de C., Lucena, C.J.P.; A Multi-Environment Multi-Agent Simulation Framework for Self-Organizing Systems. In The 10th Workshop MABS at AAMAS'09, May 2009.

[17] Giunchiglia, F., et al.: The tropos software methodology: Processes, models and diagrams. Technical Report No. 0111-20, ICT - IRST (2001)

[18] Harel, D.; On visual formalisms. Communications of the ACM, V31 I5 pp514-530, 1988.

[19] UML 2.x OMG Specification. http://www.omg.org/

[20] Wiener, N., Cybernetics or Control and Communication in the Animal and the Machine, Paris, Hermann et Cie - MIT Press, Cambridge, MA, 1948.

[21] Zambonelli, F., Jennings N. R. e Wooldridge M. Developing multiagent systems: The Gaia methodology. ACM Trans. Softw. Eng. Methodol., 12(3):317–370, 2003.