Methodology for a New Agent Architecture Based on the MVC Pattern

Yassine Gangat, Denis Payet, and Rémy Courdier

University of La Réunion, LIM
Saint-Denis, Island of La Réunion
{yassine.gangat,denis.payet,remy.courdier}@univ-reunion.fr

Abstract. In the last few years, the multiagent system's paradigm has been more and more used in various fields, specially in the simulation field (MAS). Whenever a new application came into being and has been validated by its review board, specialists usually want to reuse it, fully or partially, in order to cut down the time and price of developing similar application.

But this reuse is not as simple as expected. In a previous article, we proposed the DOM modeling to tackle modeling difficulties which arise in a complex system. However this solution has its limits as we will develop here. In this paper, we define a more complete agent modeling, based on the MVC design pattern, in order to to push back these limits.

Keywords: multi-agent, behaviors, MVC, aMVC, design pattern.

1 Introduction

1.1 Context

In the last few years, the multiagent system's (MAS) paradigm has been more and more used in various fields, especially in the simulation field. While some applications are used for pedagogic purposes, others are made in order to provide decision-support tool, e-commerce application, etc. which implies a wide range of complexity's level. Like any new paradigm, the MAS's wide-spreading requires new models, new methodologies and new softwares to support development engineers with robust and reliable applications.

Whenever a new application came into being and has been validated by its review board, specialists usually want to reuse it, fully or partially, in order to cut down the time and price of developing similar application. But this reuse is not as simple as expected.

The idea of Dynamic-Oriented Modeling (DOM) [1] was also born of this desire of model's reusing. In that previous article we proposed the DOM modeling to tackle modeling difficulties which arise in a complex system. These difficulties mainly consist in the fact that multiagents systems are becoming more and more hard to model due to the complexity of the system studied. After having implemented DOM on multiple application, we realized that DOM was not enough to

A. Ramsay and G. Agre (Eds.): AIMSA 2012, LNAI 7557, pp. 230–239, 2012.
© Springer-Verlag Berlin Heidelberg 2012

overcome all of these difficulties, especially because it doesn't care about agents' behaviors.

Indeed, to complete this one, in this paper, we study some example and propose a new behavioral model based on the well-attested MVC model. It is an original reuse of this well know pattern to obtain a new modelization approach for modularizing not only the development of environment but the development of agents as well.

As such, we first start by breaking up an agent according to some axis. Then we will recompose this splitted agent in a MVC-like pattern (aMVC) to define our new modeling proposal: the Multi-Behaviors Modelization.

Lastly, we conclude by giving a few perspectives of this work.

2 From the Breaking Up of an Agent...

This new approach called Multi-Behaviors Modelization is based on the splitting of the agent into severals pieces. The first step has already been presented in a previous paper [1]. After that first one, the next steps are phases that will allow us to build an MVC based agent.

2.1 Splitting of the Agent According to Dynamics

When we say "splitting of the agent according to dynamics", we mean environment splitting reflected on the agents. In a previous paper [1], we have presented a modeling method DOM (Dynamic-Oriented Modeling) based on dynamics, where we started with a basic agent (*c.f.* Figure 1a).

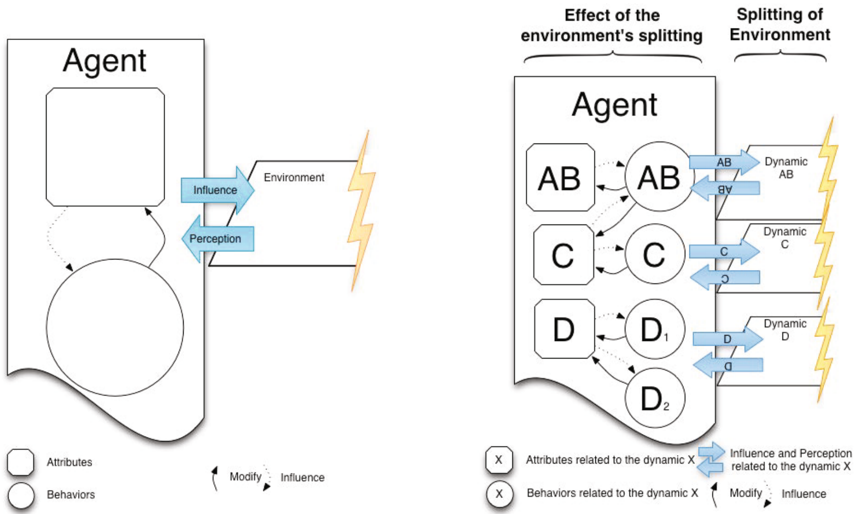
A "**dynamic**" is an association of a set of activities that participate in a major characteristic of a complex problem.

The aim of this modeling method is to break down a complex problem into some less complex parts (*i.e.* dynamics), using the environment (which is the location where agents evolve) as the coupling element for these dynamics in the Figure 1b.

In a few words, DOM is based on the integration of several layers called Mono-Dynamic Model (MDM), where each layer is related to a specific activity (such as population evolution, or flow of energy), into a multi-MDM model.

Let's take a little fictive illustrating study case: We want to make an Agent-Based Simulation (ABS) about wolves (and other animals). We will focus ourselves on wolves. A wolf agent will have some states, behaviors and interactors with environments. The first split we are going to make will through dynamic. We can identify two majors characteristics in this problem: the "individual dynamic" (that will include everything related to the individual such as its emotion, health, age, *etc.*) and the "team dynamic" (that will include everything related to the pack of wolves such as its rank, hunting, migrating, *etc.*).

This DOM methodology has already been applied in previous project such as DS [2] and EDMMAS [3]. The feedback we had on this application showed us that using DOM was a good choice. Indeed it enabled us to easily reuse an "old" simulation and build a new one from it.



(a) Basic agent

(b) Splitting of the agent according to dynamics and its effect on the agent

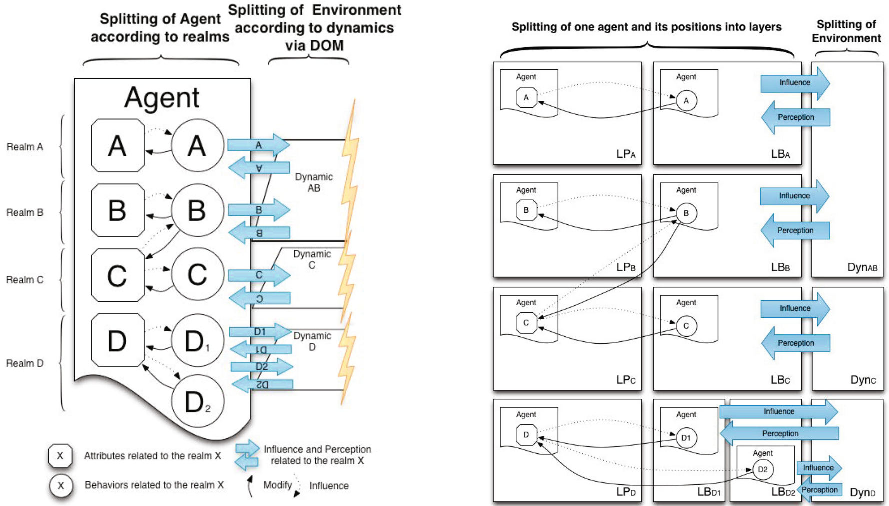
Fig. 1. From Basic Agent to the first splitting step

But the DOM methodology, based on the environment splitting, was not enough to fully apprehend modelization of complex system. We are facing some underlying problems that are general to every case of reusability. Whenever someone wants to reuse a MAS model, especially its agents, he will breast the problem of agents' behaviors. Despite the fact that dynamics had been separated, having the agents' states and its behaviors at the same level doesn't facilitate the reusability.

Indeed, DOM does not take into account the agents' behaviors, which is a critical point (as we can see in [4]) if we want to improve the reusability of our model. In this previous article, we presented a collaborative method and a NetLogo prototype focused on green turtles in the South-West Indian Ocean and we showed that each expert has different ways of modeling the turtles according to its interactions with the wind, the surface temperature, the stream, *etc.* Thus, because DOM concentrates its methodology on the environment and cannot tackle the problem of agents' behaviors, we also need to consider a methodology for the agents' modeling, especially its behaviors.

2.2 Splitting of the Agent According to Realms

In order to detach ourselves from ambiguous words, we will introduce the world "**realm**" which means area of behavioral expertise. Each set (states and behaviors) can be split according to the "realm" to which they are referring (*i.e.* in the Figure 2a the A realm, the B realm, the C realm where C is a bit related to B, and the D realm with two different behaviors).



(a) Splitting of the agent according to dynamics and realms

(b) Splitting of the agent according to its three components, realms and dynamics

Fig. 2. The second and third splitting steps

The "inside" of the agent in the Figure 2a shows us a certain amount of subsets (that could be really huge according to the complexity of the system) that are linked together. Our goal is precisely to organize everything to ease its conception.

The difference between realm and dynamic is that a dynamic can be composed by one or more realms. For example, the "social"'s realm and the "position"'s realm is part of the "communication"'s dynamic. Another example, the dynamic of "energy evolution" is composed by three realms : production of energy by plants, consumption of energy by residential houses and consumption by factories.

In our previous study case, we would be able to split the agent according to three realms: Emotional Wolf realm, Survival Wolf realm (both included in the "individual dynamic") and the Social Wolf realm (included in the "team dynamic"). We can notice, that realms of other animals can be included in the same dynamics (*e.g.* Survival Moose realm in the "individual dynamic").

2.3 Splitting of the Agent According to Its Three Components

An agent can be identified by three components:

- The agent's states, which contains its attributes.
- The agent's behaviors, which organizes all the actions it can undertake (decisional process).

– The agent's interactors, which allows interaction (influence and perception) with the environment.

In this new approach, we are taking an extra step in our initial DOM partition, in order to separate behaviors to free experts from behaviors unfamiliar to them. In this structure, we will define the whole "world" as an aggregation of several layers of physiognomy (LP), several layers of behaviors (LB) and interactors (*Influence* and *Perception*).

From this splitting by realms, we then can also split every agents and put them in differents physiognomies and behaviors layers.

If we take the same example, it will be as follows (in the Figure 2b):

– 4 Layers of physiognomies LP_A, LP_B, LP_C and LP_D.
– 5 Layers of behaviors LB_A, LB_B, LB_C, LB_{D1} and LB_{D2}.
– 10 Iteractors for each agent: 5 for *Influence* and 5 for *Perception*

This cutting of the agent into realms allows us to complete the one obtained by DOM [1] in term of dynamics. In this example, if we supposed that the splitting will result into three dynamics (Dyn_{AB}, Dyn_C and Dyn_D), their relation will be like in Figure 2b. In this figure, we can see the differents layers of the system, illustrated with the splitting of one agent. If we have hundreds of agents of the same kind, the same layers will be shared amongst them.

Note: In Figure 2b, we choose to simplify by showing the splitting of only one agent; but in fact, every agent will be split by the same realm and sent to the adequate layers. It could be represented as in Figure 3a. The advantage of this technique is that usually in a complex system, there are many agents that can be categorized by "kind". Each "kind" will be defined by the same set of behaviors, *e.g.* behaviors of an Omega wolf will be the same for every Omega wolf. By taking behaviors away from the agent's state, we are then able to factorize behaviors and reduce the complexity of the model.

Layers of Behaviors. One behavior's layer consists in definition of agents behaviors related to one realm.

A layer of behaviors is not supposed to contain the whole behavior of the agent, but its behavior related to one realm of the complex system. It could be defined by known method such as: logic definition, hard-coded definition, color-coded definition, formularized definition, Turing machine definition, tabular definition, matrix definition, *etc.*

These are only few examples, but it could use and combine a wide set of modelization methods, depending on the way the expert wants to model in his system.

Layers of Physiognomies. The physiognomy's layers is in fact a set of dynamic states related to one particular field (*c.f.* Figure 1a and Figure 2b). We used the word "**physiognomy**" in order to express the "character" or "personality" of the agents (its states and some internal laws related to this field) but

not what is usually called "body", because we did not incorporate the capacity of interaction here (which is usually associated to the body).

As you can see, an agent's states related to a particular realm (such as C) can influence and be modified by behaviors of the same agents, but related to an another realm (such as B). Moreover, a unique layer of physiognomy (*i.e.* LP_D) can be linked to two or more layers of behaviors (such as LB_{D1} and LB_{D2}).

In our previous study case, let's try this three components' split on the Survival Wolf realm. It would result in a set of physiognomies (such as its health, stamina, age, hunger, *etc.*), behaviors (attacking, calling for help, patroling, *etc.*) and interactors (walking, running, biting, *etc.*). Wolves are usually not hunting when they are hungry, they usually call their pack in order to organize an attack. This realm is therefore related to the Social Wolf realm.

3 ...to the MVC Agent

3.1 Introduction to Design Patterns

Since the introduction of patterns by Christopher Alexander in 1977-1979 in the architectural concept field, the idea of design patterns in software development started in 1987 [5] and gained popularity in 1994 after the book of [6].

Design patterns encourage reusability and can be used as "building blocks" for complex software. Several researches towards the reuse of model have been made in various fields as Software Engineering but also in Artificial Intelligence, *etc.*

Aridor and Lange's paper [7] was one of the first pioneer in applying design patterns to the MAS field. Ideas are emerging like PASSI (Process for Agent Societies Specification and Implementation) [8]. Since, several research has been done as resumed by [9], but most of the work has been focused on patterns for agent-oriented software or for the agent's interaction. Moreover, when the proposal is a pattern-based design methodology, the proposed patterns are usually homemade or specific to one domain. As stated by [10] to maximize the benefits of design patterns, they should be applied uniformly throughout the MAS research community, that would result in spreading MAS solutions and giving valuable feedback to the MAS research community.

As mentioned by [11] and [12], in order to create sets of system components needed to support highly interactive graphical software development, the MVC strategy has been chosen. Isolating components from each other as much as possible helps the application designer to understand and modify each particular unit, without having to know everything about the others.

3.2 Model-View-Controller

If we get back to the basic concept of the MVC paradigm [11], we will see that the **view** manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application. The **controller** interprets

Table 1. Example of MVC in a GUI component

Designation	`Button` in Swing
Model	`ButtonModel`
View	`ButtonUI`'s visual representation
Controller	`ButtonUI`'s handlers

the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate. Finally, the **model** manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

MVC implementation in smalltalk [11] inspired many other GUI frameworks. As an example, if we consider the `Button` class in Swing (Java [13]), the class which is used to represent a simple push button, we could see that Swing uses a variant MVC (where V and C are linked together). The `Button` class (see Table 1) is associated with a `ButtonModel` implementor for the **model**. It encapsulates the internal state of a button and defines its behaviors. `Button` class is also associated to a `ButtonUI` for its view, and possibly one or more event handlers for its controller.

Nearly all of the complex GUI elements in Swing use the component-level form of the MVC pattern for a number of excellent reasons, but the most important for us here is that it's highly reusable and it's easier to customize a component and link the components together.

3.3 Applying MVC to MAS: aMVC

If we apply this pattern to our MAS of a complex system , we will benefit most of the advantage of the MVC pattern. In order to head toward this, we need a new approach, starting from the bottom. The aim of our discussion is not only to propose a well-known design pattern, that could easily be both comprehended by experts and implemented by any developers. But we also want to underline the parallel between this design pattern and our modelization.

It's important to note that we are not talking about the software's architecture (as we can see in [14–16]) or a methodology in order to help in determining the types of agents needed to build successful MAS (such as in [17]), but about **the agent's architecture** itself. Applying MVC software's architecture would "simply" consist in using MVC in the context of software engineering, *e.g.* in a platform it will be separating visualisation of the world from the internal ABS's engine, usually through Object-Oriented Programming (OOP). Here, we want to import a methodology (MVC) existing in OOP into the world of MAS and use it at as a design methodology for the agent. In a nutshell, we are going to use an **aMVC** (agent MVC) pattern.

Table 2. One layer of an aMVC agent

Designation	Agent in one realm X
Model	its states & internal laws of X
View	its interactors with the environment related to X
Controller	its behaviors in X

We have to ascertain identity of each concept in our current model: Who is the **model** ? Who is the **view** ? Who is the **controller**? In order to do that, we will talk for the next three sections of an agent related to one particular realm.

Defining the Model. In our modelization, the states' collection mixed with internal evolution laws related to the realm (such as aging of an agent) should be the **model**. We usually tag **model** merely as a database in Software Engineering; but the **model** in MVC is both the data and the domain logic needed to manipulate the data. Thus, identifying this to be the **model** is a good solution: *LP* is a subset of dynamical data (states of agents) which evolves with time due to *LBs* and internals laws of the realms (that make the consistency of the data).

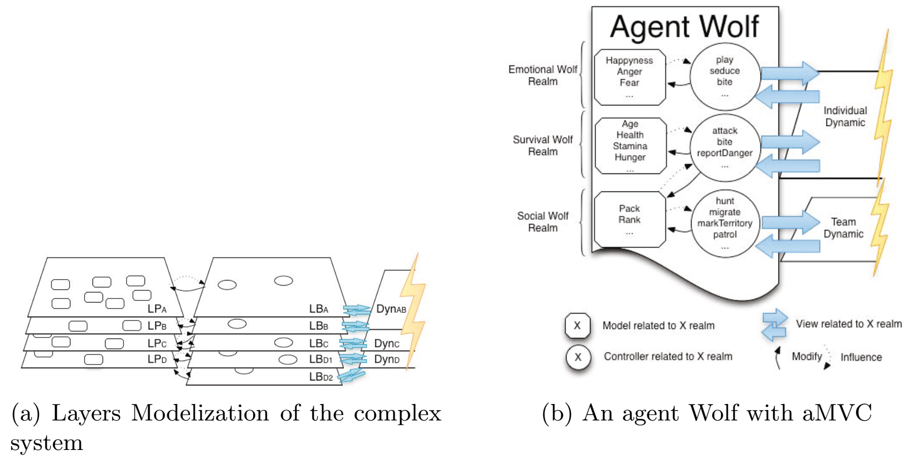
Defining the Controller. By adopting this point of view, we have identified the **controller** : the behaviors. The behaviors manage the agent's interaction with the environment and, for this, use its states: either in order to consult the states to take a decision or to influence its modification in order to memorize any experience learnings.

Defining the View. Now, last but not least, the **view** has to be defined.In other words, we can say that an agent's perception is similar to a button or a checkbox (in a graphical UI) which allows it to perceive external informations, and its influence is like a textbox or colored gauge through which the component can transmit informations (and do an influence) to the outside. The **view** of an agent is then the agent's interactors (which allow influence and perception) with the environment in a particular realm.

3.4 An Agent with aMVC

If we used the upper definition of aMVC, our agent will be a many-layered aMVC component, where each aMVC component is related to a realm. In the Table 2, we can see one layer of an agent in an aMVC form. This way, if we are taking one aMVC layer for each realm, we are then able to make a many-layered aMVC agent such as in Figure 2a.

When we compare the Figure 1a (of a basic agent) and the Figure 2a (of an aMVC agent by following the complement of methodology we proposed in this



(a) Layers Modelization of the complex system

(b) An agent Wolf with aMVC

Fig. 3. Layers Modelization and an example

paper), we could see that we found a way to organize the agent's components in a way that would ease the definition, the use and reuse of it.

In our previous study case, we will have an agent Wolf modelled according to aMVC such as in the Figure 3b.

4 Conclusion and Perspectives

This new modelization starts from the bottom (the agent) and not from the top (the system). We sliced the agent into a "**mille-feuille**" (where a layer is a realm) and then again according to its three components : physiognomy, behaviors and environment's interaction. This leads us to split also the environment into dynamic like we did before in DOM.

Due to the real splitting we would be able to give the layer to any experts, and if necessary divide the work among different experts thanks to the aMVC slicing by giving any part of the layer (M, V or C). This modelization help us in the creation of the agent. Additionally using the (a)MVC pattern's property, among other advantages, we would be able to make easy the reuse as well as the customization of any part of any layer of an agent.

This approach allows us to perceive a new field of investigation, particularly in a global level of layers' organization and the potential dynamic evolution of its interconnections, but also in aMVC: How far are the similarities between MVC and aMVC? Would we be able to apply variations of MVC to aMVC? *etc.* The study of this field will be the subject of further researches.

References

1. Payet, D., Courdier, R., Sebastien, N., Ralambondrainy, T.: Environment as support for simplification, reuse and integration of processes in spatial MAS. In: IEEE International Conference on Information Reuse Integration, pp. 127–131 (2006)
2. David, D., Payet, D., Botta, A., Lajoie, G., Manglou, S., Courdier, R.: Un couplage de dynamiques comportementales: Le modèle DS pour l'aménagement du territoire. In: JFSMA 2007, pp. 129–138 (2007)
3. Gangat, Y., Courdier, R., Payet, D.: Démonstration: Aménagement énergétique d'un territoire - une approche par simulation multi-agents. In: Journées Francophones Systèmes MultiAgents, JFSMA 2009, pp. 237–240 (2009)
4. Gangat, Y., Dalleau, M., David, D., Sebastien, N., Payet, D.: Turtles are the turtles. In: European Simulation and Modelling Conference, ESM 2010, pp. 439–442 (2010)
5. Smith, R.: Panel on design methodology. SIGPLAN Not. 23, 91–95 (1987)
6. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns. Addison Wesley (1995)
7. Aridor, Y., Lange, D.B.: Agent design patterns: elements of agent application design. In: Proceedings of the Second International Conference on Autonomous Agents, AGENTS 1998, pp. 108–115. ACM, New York (1998)
8. Cossentino, M., Burrafato, P., Lombardo, S., Sabatucci, L.: Introducing Pattern Reuse in the Design of Multi-agent Systems. In: Kowalczyk, R., Müller, J.P., Tianfield, H., Unland, R. (eds.) Agent Technology Workshops 2002. LNCS (LNAI), vol. 2592, pp. 107–120. Springer, Heidelberg (2003)
9. Klügl, F., Karlsson, L.: Towards Pattern-Oriented Design of Agent-Based Simulation Models. In: Braubach, L., van der Hoek, W., Petta, P., Pokahr, A. (eds.) MATES 2009. LNCS (LNAI), vol. 5774, pp. 41–53. Springer, Heidelberg (2009)
10. Cruz Torres, M.H., Van Beers, T., Holvoet, T.: (No) more design patterns for multi-agent systems. In: Proceedings of the Compilation of the Co-located Workshops on DSM 2011, TMC 2011, AGERE! 2011, AOOPES 2011, NEAT 2011, VMIL 2011, SPLASH 2011 Workshops, pp. 213–220. ACM, New York (2011)
11. Burbeck, S.: Applications Programming in Smalltalk-80: How to Use Model-View-Controller (MVC). Softsmarts, Inc. (1987)
12. Krasner, G.E., Pope, S.T.: A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. Journal of Object Oriented Programming 1, 26–49 (1988)
13. Stelting, S., Maassen, O.: Applied Java Patterns. Prentice Hall PTR (2002)
14. Amblard, F., Ferrand, N., Hill, D.R.C.: How a Conceptual Framework Can Help to Design Models Following Decreasing Abstraction. In: SCS-European Simulation Symposium, Marseille, France, pp. 843–847 (2001)
15. Nutaro, J., Hammonds, P.: Combining the model/view/control design pattern with the DEVS formalism to achieve rigor and reusability in distributed simulation. The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, 19–28 (2004)
16. Nguyen, T.K., Marilleau, N., Ho, T.V.: PAMS – A New Collaborative Framework for Agent-Based Simulation of Complex Systems. In: Bui, T.D., Ho, T.V., Ha, Q.T. (eds.) PRIMA 2008. LNCS (LNAI), vol. 5357, pp. 287–294. Springer, Heidelberg (2008)
17. Mahmoud, Q.H., Maamar, Z.: Applying the MVC design pattern to multi-agent systems. In: Canadian Conference on Electrical and Computer Engineering, CCECE 2006, pp. 2420–2423 (2006)