GAMA: an environment for implementing and running spatially explicit multi-agent simulations

Edouard AMOUROUX[1,2], CHU Thanh-Quang[1,2], Alain BOUCHER[1], Alexis DROGOUL[1,2]

[1] AUF-IFI, MSI, ngo 42, Ta Quang Buu, Ha Noi, Viet Nam
[2] IRD, GEODES, 32 av. H. Varagnat, 93143 Bondy Cedex, France

Edouard@amouroux.biz , ctquang@ifi.edu.vn, alain.boucher@auf.org, drogoul@mac.com

Abstract. In this paper, we introduce the GAMA (Gis & Agent-based Modelling Architecture) simulation platform, which aims at providing field experts, modellers, and computer scientists with a complete modelling and simulation development environment for building spatially explicit multi-agent simulations.

The most important requirements of spatially explicit multi-agent simulations that our platform fulfils are: (1) the ability to transparently use complex Geographical Information System (GIS) data as an environment for the agents; (2) the ability to handle a vast number of (heterogeneous) agents (3); the ability to offer a platform for automated controlled experiments (by automatically varying parameters, recording statistics, etc.); (4) the possibility to let non-computer scientists design models and interact with the agents during simulations.

While still in its implementation phase, the platform is currently used for two main applications. One is about the modelling of the spread of avian influenza in a province of North Vietnam in collaboration with CIRAD (French Agricultural Research Centre working for International Development). Its goal is to simulate the poultry value chain of a whole province using geolocalised data, and to use this to optimise a monitoring network. A second application conducted with the Institute for Marine Geology and Geophysics (VAST, Hanoi) is about using an interactive simulation for supporting decision-making during urban post-disaster situations. This application relies on geolocalised data as well, and requires facilities of interaction between users and the simulation.

Keywords: Multi-agent system, modelling and simulation, Geographical Information System.

Introduction

The use of computer modelling and simulation for the study of complex natural or social phenomena has been gaining momentum since several years. This has been the case for traditional techniques, such as mathematical approaches based on system dynamics, but even more so for the so-called Individual-Based Modelling (IBM) approaches. The first kind of models represents a system by a set of global aggregated variables, and its dynamics by equations or constraints that link these variables and express their evolution. They are interesting in that they can sometimes be solved analytically, but are very sensitive to local heterogeneities. On the contrary, IBM models are based on entities that represent individuals and their interactions in the system modelled. They can explicitly take the environment and the different actors of the system into account, with all their attributes and behaviours. They provide an interesting way for exploring, analysing and understanding situations of the emergence of global properties from the interaction of local components of the system. Finally, thanks to their fine-grained approach, these models enable the scientists to watch and follow various outputs (from global statistical results to detailed reviews of local behaviours) of the evolution from the system.

A subcategory of these IBM models concerns the ones based on "spatially explicit multi-agent systems (MAS)". Here the entities (or agents) are explicitly situated in a (possibly complex) simulated environment, in which they can perceive the information available in their neighbourhood, act upon it, and interact locally with their neighbours. They are particularly well suited for fine-grained simulations based on realistic environments such as the ones represented in GIS. These simulations are becoming more and more popular nowadays, especially in the area of decision-support systems (DSS) for social, ecological, agricultural or biological concerns. Unfortunately there are no real development environments available on the market for designing and implementing such simulations. Instead of focusing on high-level properties of the model, designers and modellers still have to deal with low-level constraints that are, moreover, remarkably similar from one project to another.

Requirements of a modern simulation platform

To obtain a generic yet useable simulation platform for the subgroup of situated agent-based model requires several issues to be addressed.

Environments

As we partly focused on the geographical data accessibility aspect, the platform has to support the main GIS file format. At the moment the most common GIS file format is that of the shape file, promoted by ArcGIS, but it is important to provide computation tools that can be adapted to handle other formats like OpenGIS.

Continuous environments like the one provided by the GIS are not the only kind of topological space in which simulated agents can behave. There are at least 2 others: discrete environments as well as networks ones that are commonly used by modellers (see [1] for extended description of multi-agent's environments).

A simulation platform must then provide sufficiently generic abstraction of the environment in order to allow modellers to manipulate them without hassle.

Agents

A modern tool should provide facilities to implement specific agent architecture but also the possibility to instantiate agents straight « out of the box » (a la NetLogo) without much need for programming. The first point requires that abstractions like the decision mechanisms, the internal state, the relations with the outer world be clearly formalised, organised and reusable within the platform. They should be offered as a well documented meta-model and an easily extensible framework.

The second one requires that at least a specifically designed language (or graphical user interface) to program agents without the complexity of a computer programming language.

The definition of environment and of an agent architecture along with a simulation management defines a **meta-model** which is a very useful guideline for both developing a new specialisation (from a framework point of view) and for the modelling process (from a user point of view).

We would now have a generic and useable platform but it still lacks ease of use and fast development tools to create a simulation from a model. To address this specific issue, the platform should at least provide a simple way to instantiate simulations from a model. To do so a common yet efficient mean is to develop a modelling language that would be automatically translate into a simulation using already defined classes.

These requirements have for main consequences that we should have a ready to be used platform with the previously discussed features and a library or framework organisation that will unbound the limits of the proposed platform when specific needs occur (depending on the modelised system, data availability, etc).

State of the art

The multi-agent simulation platforms offer is vast. However some platforms that are mature enough but lack some important features and the new platforms that are being developed to address some of these needs even still lack maturity and don't address them globally. In addition many multi-agent platforms, like Jade for instance, don't offer a good support of topological spaces or no environment at all and, consequently, have to be put aside.

Table 1. Agent-based simulation platforms review considering the previously defined requirements

	Swarm	NetLogo	Mason	RepastJ
GIS	Partial (uneasy)	Partial (raster)	No	Good (but only through Example)
Generic Environment	No	No	Yes	Yes
Generic Agent	No	No	Yes	Yes
Meta-model	Yes	Yes	No	No
Ease of use	No	Yes	No	No
Main strength	Swarm concept	Ease of use	Speed	Well developped libraries
Main weakness	Complexity	No complex model support	Complexity	Difficulty to use by modellers

Regarding simulation multi-agent platform, the first one for review (see [2] for extended review) Swarm [3] because of its historical importance. Indeed, there exists two different implementations of this platform, the original one is developed in Objective-C and the new one is in Java. They both lack real (*easy / useable*) GIS support. Although it provides a meta-model with the concept of Swarm, every agent of the simulation is both an agent and an environment for other agents, it has several drawbacks. This is a very rigid organisation and a complex one. Finally it doesn't offer any easy way of modelling, especially for non computer scientists, they would have to learn a programming language.

The second platform to review is Netlogo [4]. This is an educational platform and so it has ease of use in mind. Here there is no need to learn an obscure programming language, it offers a very easy and quick to learn language yet a quite powerful one. Although its potentiality is not as *great* as other platforms but it remains useable for many developments. The problem with Netlogo starts also with GIS support, only simple raster data can be used as an environment. To conclude on environments, only the grid has been developed and as the platform is not open source it is very complicated to develop new environments. The meta-model is also very simple consisting of only reactive agents, acting and interacting in a grid environment.

Like the well known platforms previously presented, new ones deserved to be closely reviewed. The first is MASON, its main features is its innovative design idea for execution speed. The drawback of this is the complexity to implement a model and to run a simulation. Finally it also does not support yet (MASON is a recent platform) GIS.

The last platform we reviewed extensively is the RepastJ (version 3) one. It is a bit less recent than MASON and so it already has much more developed features, especially a GIS support thanks to OpenMap. It still lacks ease of use though it has a well organised platform architecture (and good documentation).

Considering all these advantages and defects we chose to base our development of an extensible yet generic platform on RepastJ having in mind the accessibility and re-usability of the developed platform.

GAMA's general organisation

As our aim is to develop both a generic architecture for situated agent-based modelling and a ready to use platform we had the following strategy. Firstly, we chose to clearly separate the simulation process, managed in the S*imulation Kernel*, from the modelling process, managed in the *Meta-model*. Secondly, every "first class entity"("a program building block, an independent piece of software which [...] provides an abstraction or information hiding mechanism so that a module's implementation can be changed without requiring any change to other modules." in E4MAS'04) of the platform is thought to be re-definable resulting in the possibility to use GAMA as a framework and allowing developers to build their very own simulation platform considering their specific needs.

Modelling ontology

Our meta-model is thought as guidelines to build any kind of simulation from physical phenomena to social ones. To be generic we define a minimum set of needed concept to build a model and organised them in the meta-model (see fig.1). The two main concepts to consider are *Agent* and *Environment* at first.

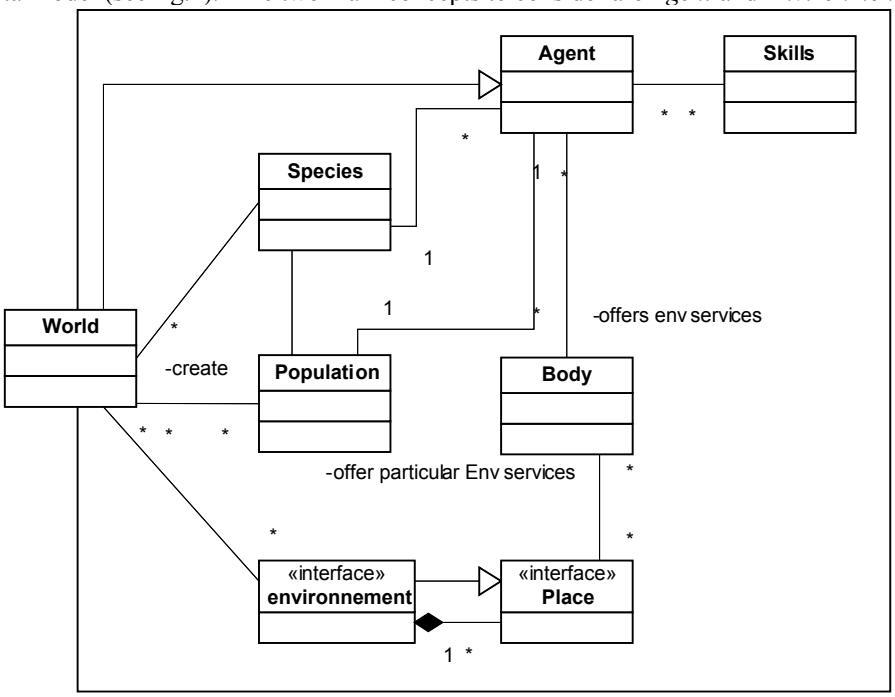

Figure 1
Meta-model: a world managing agents and environments

In GAMA what we call *Agent* only takes care of holding the internal state of the modelised entity and a decision system. Consequently we will refer to common "Agent" sense in literature with "Entity". The decision system of an Entity can be of any kind, from cellular automata, rule-based system or complex cognitive system. To have an example of a possible decision system we will define the one used in conjunction with the Gama Modelling Language (GAML) in the corresponding section. The internal state is only a set of variables and is less likely to be redefined. To catch up the common agent paradigm we defined two other concepts: the *Body* and the *Skill*.

Entities are generally localised in an *Environment*, spatial or conceptual one. This localisation and actions within an *Environment* are done thanks to the *Body* class which acts as an interface between them. The *Body* class offers the following localisation services:

- Communication and perception with the use of *Variables* (stigmergy) and signals
- Neighbourhood, which agents are around

- Movement possibilities

Using the standardised basic low-level localisation service of any environments (that will be described below), the *Body* is generic to all environments defined in the GAMA standard and is able to manage several environments at once. As an Entity can act on various environments at the same time, the *Body* will manage all the requests of the *Agent* and direct them to the relevant one.

For example a Human agent can be in a house and communicate with them, using the topological space, and at the same time would be able to communicate to other peoples through telecommunications, using a social space.

Skills are a set of possible actions for the Entity. These actions can be very different from interaction between Entities, action on the Environment or even with the GUI (e.g. reveal/hide the agent). In GAMA, generic *Skills, the Carrying* one allows an agent to carry other ones for example, those that have been already defined though new ones would be needed to fulfil specific needs depending on the modelised system. More precisely Skills embed effective Java methods that will allow the Entity to do actions defined in the model and present what kind of variables are needed or optional for the Entity to fulfil those particular actions.

To allow Entities to perceive, act and interact among them they need a substrate to do so, that is where *Environments* come into the light. "An environment can either be represented as a single monolithic system, i.e. a centralized environment, or as a set of cells assembled in a network, i.e. a distributed environment" (Ferber, J.: Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence. Addison-Wesley, ISBN 0-201-36048-9, Great Britain (1999)). In GAMA every type of *Environments* are unified thanks to the generic network organisation consisting of an aggregation of *Places. Places* are where the Agents are effectively localised and where environmental data are stored. *Places*, in coordination with the *Environment*, also offer a minimum set of low level services to the Entity's *Body* as follow:

- Store and access (perception) environmental data (including stigmergic data)
- Space topology (movement constraints) and access to neighbourhood (*Places*)
- What Entities are at the same place

There are 3 types of environment (continuous, discrete and network) to fulfil all the possible needs and we developed an implementation for each of these. The first one is a Network space, it connects *Places* without concern for any real physical space and is generally used to transcribe social relation. The second one is the Grid *Environment* consisting or arranged cells. It allows testing of models when no real field data is needed or to test a model with forged data and has the benefit of being computationally efficient. Finally, the third one is continuous space with the use of GIS. Here the *Places* are GIS objects. The three types are currently implemented in the GAMA platform. Especially the GIS one based on the OpenMap library and is ready to be used which is a great requirement as we stated before. Although, it is possible depending on specific needs to use the abstract concept of one of these three *Environments* to define a new implementation. As long as the developer follows the organisation and the *Environment*, the new implementation would be fully and smoothly integrated in the GAMA architecture (even for a monolithic *environment*).

These concepts of Entities (*Agent* + *Skills* + *Body*) and *Environment* would be enough to define an agent-based model but are not enough to do it efficiently. We added automatic mechanisms to ease model to simulation process. We did so with three concepts as follows that will instantiate the Entities and then manage the evolution of the simulation.

Generally Entities can be regrouped under a few different prototypes; we decided to formalised and propose an implementation of this mechanism (with the instantiation one).

The *Species* is such a prototype it defines both the internal states and the behaviour. Though, the behaviour is described following the decision system chosen in the *Agent*'s architecture. For example a generic human in a simple model could be defined as an *Agent* with the following characteristics:

Internal state: age, strength, speed, sex
Skills: walk on ground, swim in water, climb on trees
From this base we can build various models from prehistoric men looking for food to in contrary kids lost in an amusement park.

To instantiate or specialise this generic *Agent* we added the *Population* class. It is responsible for building a concrete *Agent* from the prototypical one. It will specialise *Variables*, the behaviour (depending on the decision system) and sets the localisation.

To go back to our first example, *the population would consist of 50% man and woman, localised in a village, with age, strength and speed distributed over statistical functions.*

As the example pointed out, it is possible to use mathematical function to distribute parameters over a space.

Finally, the *World* object is responsible for the instantiation of all of these objects (reading a parameter file) and their coherent interactions throughout the simulation. As the *World* is also an *Agent* it is possible to give it a behaviour. Actually its behaviour is to follow a predefined scenario (executes events e.g.) and to provide data about the current situation of the simulation to the GUI packages that enables the user to track the evolution of the system and eventually interact with it.

Simulation kernel

The simulation kernel is in charge of the technical processing to build a concrete simulation from a model. In fact, an agent-based model is a static object that describes a dynamic system. It consists of an *Environment*, Entities and their action on it and among them, by building all the entities of the model and letting them execute themselves it gives effective dynamicity to the model. To do so, and to be able to monitor and control the run of the simulation, we developed a simulation kernel offering the full range of essential services.

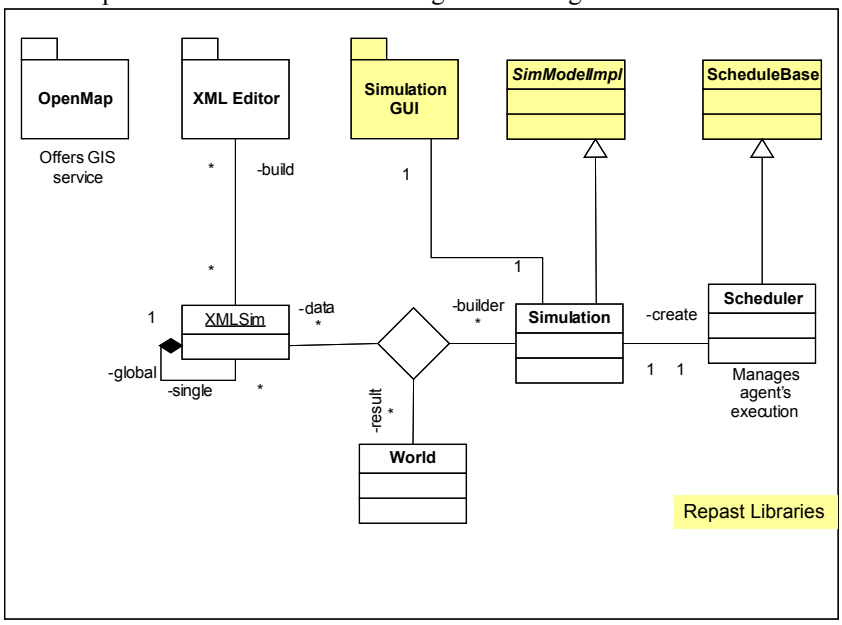

Figure 2
Simulation kernel: providing services to execute a simulation

The main class and the entry point of this kernel is the *Simulation* class (see fig. 2). It is responsible for reading the parameter file (XML-based), instantiate all the needed objects and delegating the parsing of specific XML elements to the *World* object.

The *Scheduler* is responsible for the correct execution of the simulation, it allocates time to every *Agents* to execute themselves, manages coherently newly created *Agents* and disposable one within the execution process. Clearly an Entity can be discarded from the system only when no other *Agent* is having an interaction within the first one. The newly created Entities may respect some conditions too, for example in a physical environment an Entity can not be exactly at the same place as another one.

When all this technical work is done and the simulation is ready to start, the *Simulation GUI* comes onto the scene. Its responsibility is to allow users to monitor the evolution of the system. As various needs exist, depending on the purpose of the simulation, we had to allow monitoring from a single Entity (its internal state e.g.), using individual probes, to global (or group restricted) or statistical measures, using graphs. It is also possible to track the spatial evolution of the system within the *Environment*, which a mathematical model cannot allow. For example, watch Enitites' movements and track their trajectories over time. On top of visualisation we added the possibility to dynamically affect the simulation throw interaction mechanism. It is possible to check an Entity's internal state then change values and, more importantly, its behaviour. For example, in our rescue simulation, we will capture decision process throw interaction with the simulation. Decision makers can change Entity's behaviour on the fly to conform them to their own decisions.

As you may have noticed along the description of these general mechanisms, the platform has no restriction, it is possible to build complex cognitive Agent to simple reactive ones, while useful guidelines mechanism is ready to be used. If correctly followed, these guidelines guarantee the correct operation within the platform and will allow major reusability of a previous work. Since all Entities related classes are defined and offer the full range of services needed, it is very fast to model another system thanks to the *Population* and the *Environment* interfaces and classes, possibly with different entities modelled. If you need to model similar system with different data, only the configuration file has to be modified with the new parameters. We used these guidelines to build the first complete instantiation of the platform which is ready to be used.

GAML: a scripting language for developing agents

GAML is a modelling language, inspired from EMF developed during the Manta project [5], based on XML formalism and is meant to define reactive agents. The idea behind it was to propose an easy to learn language, like the NetLogo one, but not a programming one. The simulation configuration file uses a light version of it and the *World* is programmed using it for the simulation's evolution management. This language is thought to be used with the decision system we built for our reactive agent (Entity in our formalism).

Simulation configuration

The simulation configuration format uses a light version of GAML. It defines a general formalism to parameterise all the first class entities of the meta-model (*Agents, Species, Population, Environments and World).* As *Agent, Species and Population* definitions are quite specific to the Agent architecture developed along GAML they will be defined if the following section. Consequently we will now focus on *Environment* and *World.*

Defining an E*nvironment* is type dependent but is quite simple. In the case of a network *Environment*, we only define the topology of the *Places*. Concerning the grid *Environment*, the topology is implicit (4 or 8 neighbours) so only the dimensions and the metric (cell size) has to be defined. The GIS *Environment* is a bit more complex so we will look at an example. Here we have to define a reference layer (as different layers are likely to cover different zones) plus drawing informations.

```
<GIS type="OpenMap">
                        <shapeLayers>
                                <layer prettyName="road" isRef="true">
                                        <shapeFile value="data/gis/road.shp" />
                                        <lineColor value="#000000" />
                                        <fillColor value="000000" />
                                </layer>
                                <layer prettyName="hospital" isRef="false">
                                        <shapeFile value="data/gis/hospital.shp" />
                                        <lineColor value="#000000" />
                                        <fillColor value="00FF00" />
                                </layer>
                        </shapeLayers>
</GIS>
```

World which is the interface between the Simulation kernel and the Modelling ontology has also responsibility to follow a predefined scenario and to track the evolution of the simulation. To do so the *World* is defined like an *Agent* to produce event in the simulation and to provide facilities to track any evolution of the simulation.

```
<event>
      <do action=”earthquake”>
      <arg name=”magnitude” value=” rnd * 10”/>
      </do>
</event>
```

As you can see in the example, an event is just a global action on the simulated system and as you will see in the next section it relies on the *Skill* (*World* specific ones) mechanism as *World* is an *Agent,* though a specific one.

GAML and its associated agent architecture

GAML is responsible for enabling the modeller to describe all the characteristics of the Entities. Concretely, GAML will offer a description formalism for the internal state of the Entity and for its behaviour also based on the following decision system. As described in the meta-model the *Species* is the prototype of Entities and so this is where it will be done in GAML.

- *Species*

At each timestep the *Agent* selects which *Task* is the best one depending on the internal state and its perception of the environment. A *Task* is a sequence of actions and holds parameters. The Actions are implemented (and described) in a *Skill* class which is referenced by the *Species*. Here is an example from the Around application to have a clear view :

```
<task name="driveVictim">
        <priority if="!empty & !fireHere" value="20" else="0"/>
                <duration max="2 hours" />
                        <repeat action="goto">
                                <arg name="target" value="hospital" />
                                <arg name="speed" value="150 kmh" />
                        </repeat>
                <do action="drop">
                </do>
</task>
```

This *Task* makes the Entity (an ambulance for instance) go to a hospital then to deliver its victim there. The “goto” action is a repetitive one and will be executed as long as the entity has not reached the hospital (as defined in the “moveable” *Skill* class using the compulsory parameter “target”). The “do” is a non-repetititve action to “drop” (defined in the “carrying” *Skill*). As you can see this task will be executed if the Entity actually carry a victim (internal state) and there is no fire here (perception).

Another way to act for the Entity is reflexes. They are simple non repetitive action that are instantaneous action, an example could be :

```
<reflex if="health < 1">
        <do action="die" />
</reflex>
```

Here if a victim's health goes below 1, the “die” action would then take care of getting the agent out of the simulation. These 2 examples show an Entity acting in its environment but interactions among Entities is addressed through communication which is handled specifically as follow.

There are two main ways of communicating in our agent-based simulation platform, a basic message paradigm and the stigmergy. Here is an example for message:

```
<ask target=”parent”>
        <do action=”prepareFood”/>
</ask>
```

Stigmergy relies on signals landed in the *Environment* which act as a shared medium and manages propagation of these signals. Like other environmental data those signals are used thanks to the *Body*.

- *Population*

Here we defined the instantiation of Entities from the *Species*. We define groups of Entities sharing the same behaviour and localisation process, here is an example:

```
<group species=”ambulance” size=”10”>
        <subgroup size =”5”>
        <location =”hospital”/>
        </subgroup>
        <subgroup size =”5”>
        <location =”road”/>
        <int name=”fuel” value=”rnd * 100”/>
        </subgroup>
```

</group

Here we have 2 groups of ambulances, some starting in the hospital and others on the road with a random quantity of fuel (by default thanks to the species the first ones would have 100% of their fuel).

Though this modelling language is more accessible than a programming language, a graphical editor is currently under development to further ease the modelling process especially for field specialists. This editor allows the "simulationist" to focus on modelling by managing low level coherency.

Applications

The development of GAMA has then been conducted in parallel with the implementation on the platform of four complex models with very different requirements, enabling us to make a clear distinction between generic structures and services and application-specific needs.

The first model concerns the simulation of the spread of avian influenza in North Vietnam [6]. It is built in collaboration with epidemiologists and veterinarians from the CIRAD. Contrary to mathematical models, which are not designed to represent all the complexity and uncertainty of the real system, we aim at providing a model with realistic and detailed mechanisms. This implies the following needs: the ability to use field data stored in a GIS, to represent heterogeneous agents (domestic and wild birds, human and their different roles, etc) and to track the evolution of the simulation from as many points of view as needed in order to infer knowledge about the real system from the outputs of the simulation. Specifically, when the experts will declare the simulation realistic enough, it will be used as a "virtual environment" to test and optimise the deployment of a targeted monitoring network.

The second model, part of a larger project called "AROUND" (Autonomous Robots for Observation of Urban Network after Disasters), concerns the simulation and optimization of urban disaster rescue operations. It is built in conjunction with geographers and rescue experts from the VAST (see for instance [7]), and specifically targets post-earthquakes situations in Hanoi city. As in RoboCup Rescue [8], realistic aspects of the disaster are taken into account: fire, housing and building damages, road disruptions, status of victims, localization of rescue centres, etc. The search and rescue operations are implemented through heterogeneous agents that represent firemen, ambulances, and policemen with realistic individual behaviours. The aim of the project is to learn the global strategy used by rescue experts in such situations in order to provide an efficient decision support system in case of real disasters. The learning process consists in letting the experts play with the simulation, alter the individual behaviours and decisions of the agents, and in inferring, by successive proposals, trials and validations, realistic decision rules. This application requires a high level of interactivity with users and the use of very precise field data.

The third model is about the simulation of daily household activities in order to test the effect of the introduction of in-house services and appliances. It is built in collaboration with ergonomists of EDF (Electricité de France), who provide real-life scenarios (translated from video shootings of French families). Each member of the household is represented by an agent, which can either play its behaviour as stated in the scenario or be controlled by an end-user and learn its behaviour by imitation. This participatory process (detailed in [9]) requires a high level of interactivity and a precise description of the environment surrounding the agents.

Finally, the fourth model is a reimplementation of the SAMBA model [10], originally implemented in the CORMAS environment [7]. SAMBA uses multi-agent modelling for testing hypotheses about agricultural dynamics and to better understand individual decision making and its consequences on agricultural dynamics and land use change in the Bac Kan province. The original SAMBA model was limited to the scale of a village; its reimplementation in GAMA aims at adopting a multi-scale approach by taking into account behaviours from other actors: local authorities, province authorities, inter-villages trade possibilities, changes of government policies, etc. The development of GAMA has then been conducted in parallel with the implementation on the platform of four complex models with very different requirements, enabling us to make a clear distinction between generic structures and services and application-specific needs.

Conclusion

The GAMA platform is still under development but already offers most of the requirements we described. GIS support is addressed thanks to the use of OpenMap while a generic environment support as well as specific ones is implemented too. The agent architecture separating the internal state + decision system from the localisation one and from the action one represents a good guideline to implement a specific agent architecture while it is very generic. These 2 features result in a precise and generic meta-model.

From a more “end user” point of view, we added several facilities to use the platform for modelling and simulation with no programming need. GAML and the reactive agent architecture offers a ready to be use which is very accessible to non computer modellers with a minimum learning requirement. This architecture is not mandatory and the framework organisation allows re-using of previously defined model. Even simulation can be integrated without too much hassle in the framework.

The choice to base GAMA on open-source libraries like OpenMap and RePast allows us to benefit from their future enhancements while easily reusing and adapting most of the tools (visualization and manipulation tools, for instance) they already provide.

Finally Repast is going to evolve soon, we will have to review the new platform and decide whether to use or not for GAMA. As we used Repast as a set of libraries, this would not be too complex.

Acknowledgements

We are very grateful to our partners, Stéphanie Desvaux (CIRAD - Hanoi), Nguyen Hong Phuong (VAST – Hanoi), Nicolas Sabouret (LIP6 – Paris, France), Yvon Haradji (EDF R&D – Clamart, France) and Jean-Christophe Castella (IRD – Montpellier, France) for their advices on the development of the platform. This work is supported by a 3-years PhD grant from the IRD, an international volunteer grant from the French Ministry of Foreign Affairs, and a research grant from the ICT-Asia Programme.

Bibliography

1. Weyns, D., Van Dyke Parunak, H., Michel, F, Holvoet, T., Ferber, J.: Environments for Multiagent Systems, State-of-the-Art and Research Challenges (2005)
2. Railsback, S.F.,Lytinen, S.L., Jackson, S.K.: Agent-based Simulation Platforms: Review and Development Recommendations. In Simulation (2007).
3. Minar N, Burkhart R, Langton C, Askenazi M. The Swarm simulation system: A toolkit for building multi-agent simulations. Santa Fe Institute (1996)
4. Tisue, S., Wiensky, U.: Netlogo: A Simple Environment for Modelling Complexity. In International Conference on Complex Systems (2004)
5. Drogoul, A., Ferber, J.: Multi-Agent Simulation as a Tool for Modelling Societies: Application to Social Differentiation in Ant Colonies (1992)
6. Desvaux S. et al. : HPAI Surveillance Programme in Cambodia: Results and Perspectives. In OIE/FAO international Conference on Avian Influenza. (2005)
7. Nguyen-Hung, P.: Decision support systems applied to earthquake and tsunami risk assessment and loss mitigation.
8. Kitano, H.: RoboCup Rescue: A Grand Challenge for Multi-Agent Systems. In International Conference on Multi-Agent Systems. (2000)
9. Sempé, F., Nguyen-Duc, M., Boucher, A., Drogoul A.: An Artificial Maieutic Approach for Eliciting Expert’s Knowledge in Multi-Agent Simulation. In Proceedings of MABS 2005. (2006)
10. Castella, J.C., Boissau, S., Tran-Ngoc T., Dang-Dinh Q.: Agrarian transition and lowland-upland interactions in mountain areas in northern Vietnam: Application of a multi-agent simulation model. In Agricultural system. (1986)
11. Bousquet, F., Bakam, I., Proton, H., Le Page, C.: Cormas: common-pool resources and multi-agent Systems. In Lecture Notes in Artificial Intelligence. (1998)