RAFALE-SP: a methodology to design and simulate geographical mobility

Nicolas Marilleau*, Christophe Lang, Pascal Chatonnay**, Laurent Philippe****

* *UMI 209 UMMISCO (IRD-UPMC),*
32 rue Henri Varagnat
93143 Bondy Cedex
nicolas.marilleau@ird.fr

** *Université de Franche-Comté*
LIFC, 16 route de Gray,
25000 Besançon

Abstract.
Complex processes, as geographical mobilities, cannot be always modeled by mathematical formulas or probability laws due to the number of parameters that characterize their behavior. For this reason, simulation is generally used to study movements in these systems. Multi-Agent Systems (MAS) are an interesting and intuitive way to model such complex systems as they divide the whole problem into simple ones. However, in this domain, there is a lack of a global methodology that supports scientists from the design to the implementation of a simulator. In this article we present the RAFALE-SP methodology that helps out the developer in the construction of spatial mobility models and in their simulation. The methodology is implemented in a framework that is composed of an oriented *meta-model to describe spatial mobilities and an* oriented *toolkit to implement mobility simulators. This paper gives a complete description of the methodology and its concepts then it presents how to use it for the implementation of a simulator.*

Keywords: *complex systems, multi-agent system modeling, distributed simulations*

1. Introduction

Movement is something totally natural for everyone. However, studying movements is difficult when we consider real systems com-

posed of a large number of individuals (human or not). This complexity comes from the independence of the individuals and their autonomous behavior. Multi-Agent System (MAS) is an interesting and intuitive way to model such complex systems. The reason is that they use a decentralized approach to divide a complex problem into simple ones [JP97], i.e. this paradigm allows to represent a complex system by interconnecting more simple autonomous entities: agents. The shortest path algorithm based on ant colonies [BG92] is such an example where agents are used to find an optimal solution. So, creating an ABM (Agent Based Model) to study dynamics of a complex system requires to consider individuals (agent) and movements, at the natural level: each agent works to achieve its tasks and interacts with others to solve the global problem.

Some generic frameworks are available to model and simulate complex systems using an agent based approach. But none of them includes a methodology that supports scientists from the modeling stage to the simulator development. Most of the frameworks are designed to develop without formal modeling. We advocate however for the use of methodologies and formal models in the design of large simulators. Modelling is seldom used in the design of multi-agent simulators compared to the software development context where the use of methodologies, models and formalisms is common and widely accepted. They are numerous advantages for using this approach before starting the development of a simulator: changes done afterward will be simplified, when correctly specified parts of a simulator may be re-used in another one and, last, validation tools may be used to assist in the development and to check properties.

The contribution presented in this paper is a methodology called RAFALE-SP [1]. The goal of this methodology is to help scientists with the definition of spatial mobility models and their simulation. It is composed of a *oriented* meta-model [2] to describe spatial mobilities and a *oriented* toolkit to implement mobility simulators. The meta-model and the toolkit are generic because they can be used to model and simu-

1. Reflexion, Analysis, Formalization of Agents Localized on an Environment to Simulate Peregrination

2. A meta-model is a tool which intends to create models [OMG02]

late various complex systems. Yet, they are *oriented* in a sense that the goal is specific to the mobility area. This methodology can be applied to a wide span of problems as the study of population dynamics in a town, the simulation of urban traffic or the validation of algorithms for robot motions in a virtual environment. The RAFALE-SP methodology is implemented in a framework. This framework helps users with the description of mobiles, environments and movements and it provides tools and libraries to generate simulators.

The aim of this article is to present the method we established to help scientists with the modeling and simulating process of complex systems. In the second part, we define the mobility context and we present related works and their characteristics. Then, we introduce the RAFALE-SP methodology, the proposed concepts and two use cases that illustrate the methodology application all along the paper. In the fourth part, we show how environments and motions are described to create the mobility model. The fifth part presents the toolkit that implements the methodology. Finally, we compare the methodology to other approaches before concluding.

2. Related works in the mobility simulation domain

Simulating physical phenomena or behaviors is mandatory to understand the evolution of complex systems. The simulation approach is composed of several steps: choose an adapted tool, model, implement and execute. Depending on the complexity of the system, different techniques may be used as mathematic or computer science. Mathematics based representations of phenomena or behaviors lay on continuous deterministic models [DLV02]. The main interest of those models is the possibility to compute an exact resolution of the problem. However, those models are limited: on the one hand only a few number of variables may be used in the model if we want a solution and, on the other hand, the definition of a mathematical model is not simple and not always possible. So, most of the research activities on simulation use discrete event models [ZPK00]. Developing a simulator is however a complex, time consuming task and, most of the time, a simulator cannot be reused after it has given the expected results. For this reason,

the simulation research community has develop concepts, methodologies and tools to help in the design and development of simulators. The agent paradigm is one of them.

As said in the introduction, the agent concept is a powerful support to simulate mobility in systems composed of individuals. Several researches aim at creating generic tools based on agent paradigm to help users that want to create models or/and softwares. We can divide these researches into three levels: (i) methodologies (*GAIA* [WJK00], *TROPOS* [BPG+04] or *Adelfe* [PBG03]), (ii) languages (*AUML* [BMO01], *AML* [CTC05], *Ploom* [Fer90]), (iii) and toolkits (*MadKit* [GFM00], *MASON* [LCRPS04] or *Repast* [NNV06]).

These tools were created according to different approaches. Many tools come from Object-Oriented (OO) tools as *AUML*. Others are created independently of OO (e.g. *TROPOS*).

Each methodology, language and toolkit contains specific characteristics. They do not share the same representation for agent and environment. For instance, the agent language associated with *GAIA* describes environments with variables whereas *AML* represents environments with a set of objects that can be accessed by agents. Due to their specificities, frameworks do not have the same goal.

Note that, depending on the methodology, language or toolkit abilities, a tool may be more efficient in a particular domain [BLPM02]. A generic tool can be used in different situations. It is however less efficient than a tool that is designed for a specific system.

Each agent oriented tool is created to reach a specific goal (creating a software, simulating motion, etc.). Due to their objective, tools use different concepts like role, perception or mobility. Mobility studies are often achieved by creating models and software in order to design and simulate different kinds of trips. So, agent based tools like AUML allow mobility description. The meaning of the mobility concept is however different depending on the considered multi-agent tool. Some of them, like *AUML* or *GrassHopper* [BBCM00], associate mobility with *mobile computing* i.e. agents moving from one computer to another. Others associate it with functions to simulate motions. *Repast* permits simulation of agent journeys in a spatial environment. The work presented here fo-

cuses on *simulated mobilities* rather than on *real mobilities* according to the definition of motions that are proposed in Multi-Agent frameworks.

In the literature, agent-based simulation tools which aim at simulating mobility, are either agent centered frameworks or environment centered frameworks [Fou05]. In the first case, agents are fully described and environments are less realistic. On the opposite, in the second case, agents are simpler and they evolve on a accurate virtual world. We have created a new framework to fill the gap between these approaches. It allows us to simulate intelligent mobiles (e.g. citizens) that move on a complex environment (e.g. a city with its roads, shops, restaurants and so on).

3. A methodology to design and simulate complex systems

The goal of this section is to give a global description of the RAFALE-SP methodology. After justifying the approach, we define the concepts used before presenting the methodology. Then, we show how motion is described in our proposition.

3.1. *Methodology justification*

Two questions are tackled in this part: the methodology and the benefits expected form this methodology.

The work presented in this paper concern both a methodology and a framework. What is for us the distinction between these two concepts? We name methodology the approach proposed to design simulators. It is based on a progression from the model to the implementation and on several concepts that have been identified as central in the design of mobility simulators. So this is a set of working methods that helps simulator designers to develop a simulator to study a context.

We claim that this methodology is generic: it may be used with different frameworks to develop simulators. We believe further that this approach can be used for the design of other simulators than mobility simulator but we do not have results to prove it. The framework part is composed of the languages adaptation or definitions and of libraries

that we propose to help users with the methodology progression. We also present the framework in this paper.

The benefits expected from the methodology are the same as for all software engineering based designs: quality of design and implementation, evolution ability for the simulators, software reuse thanks to the modular approach, etc. In the particular case of our methodology we also plan to use the models to validate parts of the design. All these benefits leads to shorten the development of a simulator, in particular when the developer have complete mastery of the methodology and of the framework. Scientists that study mobilities and thus frequently develop simulators will benefit from the proposed methodology.

3.2. *Manipulated Concepts*

To allow the representation of autonomous individuals, each mobile is designed by a pro-active and social agent integrating human characteristics. This choice was mandatory to support the description of mobiles that are able of high level decision capacities. This is the case for the simulation of human dynamics in an urban context for instance. However, who can do more can do less and the representation of mobiles with low decision capacities is still possible. In the domain, many social agent architectures have been proposed [BH02, CBF03, ML09]. Major of them are based on BDI approach [Dav96]. VON-BDI [3] agents [BH02], an extension of BDI agents, are inspiration sources to response to our needs because these concepts were created to represent humans. Nevertheless, the BDI logic is not kept in RAFALE-SP. Thus, in RAFALE-SP, agents are based on VON-BDI architecture but not based on VON-BDI internal functioning.

As in most mobility simulation platforms, agents are located on a virtual space, the environment, which represents a real area, for example a town. So, a motion can be defined by a position change on this environment. This change is the result of a complex computation based on the agent mental state, its perceptions, its physical abilities and its internal behavior rules.

3. *Value*, *Obligation*, *Norm-Belief*, *Desire*, *Intention*

In this environment, a reference frame must be chosen in order to determine an unique location for each agent. An environment may be defined as a cellular automaton, a graph or dimensional spaces [TDZ08]. The frame of reference will determine mobilities that can be observed during a simulation, so its choice must be considered. Many considerations as the study goal or the accuracy of a simulation, need to be taken into account for its definition.

To be generic, areas studied using the methodology may be complex. Motions are constrained by space topology (e.g. streets define the places where cars move) and laws applied to the area (e.g. the highway code). As a consequence, mobile abilities are limited because they cannot move from a point to everywhere without having an intermediate location. For this reason, RAFALE-SP associates a set of motion capacities to each agent. This capacity determines actions and movements that an agent is able to do and represents the physical abilities that the studied mobiles have in the reality. For example, we can define walking and running aptitudes for an agent representing a human.

RAFALE-SP represents real mobility by two major elements: the *Agent* and its *Environment*. In addition, *Interaction* between agents and *Organization* (e.g. family, firm) are considered in our framework. In fact, RAFALE-SP is inspired by a multi-agents approach called Vowels [Dem03].

One of the issues encountered when designing a MAS is the granularity of the system representation, i.e. what must be represented by an agent to get realistic results and what must be put in the environment. The well known tradeoff is that the more we implement as agents the more realistic are the results but also the slower is the simulator. The main reason is that more agents leads to more computing power consumption, due to their dynamic behavior. In the RAFALE-SP method, we make the assumption that fine grained mobilities are modeled into individuals and the whole system dynamics are described into the environment. It means that studied mobilities can be described at a microscopic level and other dynamics of a system at a macroscopic level. Thus we can limit the execution cost of a simulator by describing parts of the mobilities in the environment.

3.3. *Overview of the RAFALE-SP methodology*

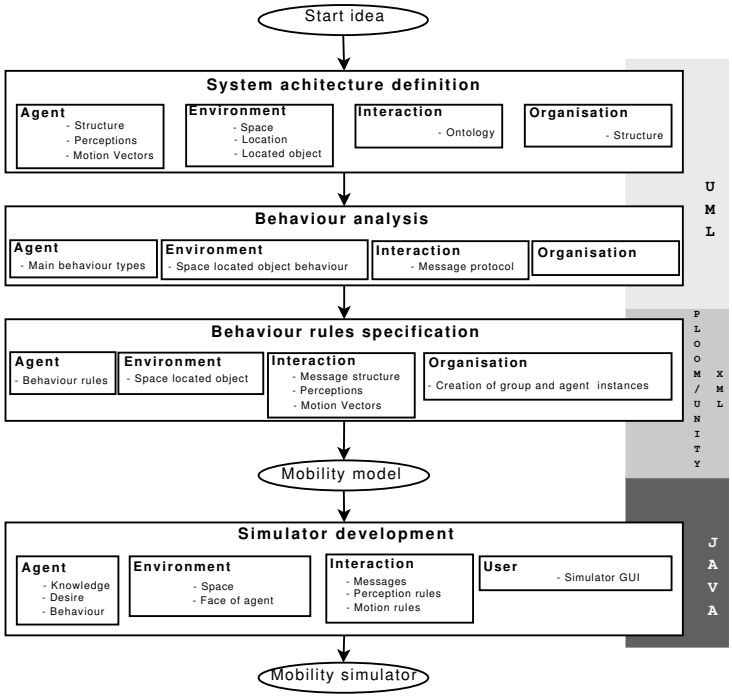

Figure 1: Steps of simulator creation in the RAFALE-SP methodology

Traditional approaches used by software engineering have already proven their efficiency in modeling complex applications. They are, unfortunately, seldom used in simulator design. RAFALE-SP and its associated methodology take advantage of these techniques. Figure 1 illustrates the steps defined in the methodology to realize a simulator. First of all, a specification document must be written. It describes, in a natural language, the system to be studied. The aim of this first step is to give a global view on the mobiles that we want to describe. So, this specification is an early document that is necessary clearly identify of what we will model.

Then, four steps are needed to obtain a functional simulator:

1) *The architectural description* represents how a system is structured. It describes, for example, the structure of agent knowledge and how mobiles perceive their environment. Moreover it shows how mobiles move on the space. UML diagrams, like class diagrams, are used to achieve this step. These diagrams are a first step before a more accurate model. They are very useful during the development phase for communication purpose. Thus, they are a good medium to globally understand the system we want to model.

2) *The behavior analysis* defines main agent behavior types. In another words, we declare different types of agent acts (e.g. move or plan a path). At this stage, we use diagrams like state-charts.

3) *The specification* describes the system with accuracy. It intends to describe mobile behavior rules. It defines efficiently how mobiles move and how agents observe their environment. Few specifications are necessary to complete this step. These specifications use Ploom-Unity [Mar05], a language which take advantage of *Ploom* [Fer90] and *Mobile Unity* [GCP03]. At this stage, the *XML* meta-language is also used to describe the structure of the messages exchanged by agents.

4) *The development* aims at implementing mobiles, their environment and graphical interface. This implementation is facilitated by a toolkit based on the meta-model structure. This library is composed of classes which are abstractions of desires, beliefs, and viewed elements.

Our methodology uses several formalisms - languages and meta-languages - as UML, Ploom-Unity and XML to model the agents and their environment. These formalisms are introduced to easier the learning and the correct use of the methodolgy. We mainly chose them for their expressiveness and for their wide acceptance in the development community so that we can take advantage of a large panel of tools, documentation, examples, etc. The ease of use also implies a better efficient in the design and implementation process.

In some cases, an adaption is mandatory to specialize the language to the specific context. For instance, we chose UML because it is well adapted to our needs and widely used but we specialized it to describe geographical mobility. In other cases, we had to design a new language to correctly implement the methodology concepts. Thus, we created the Ploom-Unity language as existing languages are not generic enough

to describe dynamics that take place on any territory. For example, ELMS [OBCdRC05] describes an area by a regular grid while our approach allows to design various types of spaces as grids, graphs or other topologies. Last, the use of the XML meta-language is obvious for all the cases where a formalism must be defined to instantiate structures.

3.4. *Use cases*

To illustrate the interest and the use of the RAFALE-SP methodology, we present here two use cases on which we have used the RAFALE-SP methodology and that come from different research area: the MIRO [4] project, in geography, Microbes, in soil sciences.

The goal of the MIRO [BCL+05] project is to model and simulate daily journeys of Dijon (town in the east of France) inhabitants. From a poll which collects data about Dijon inhabitant activities, a first analysis is done by geographers. Then, we have established an agent based model and developed a simulator by following RAFALE-SP methodology. Most of the concepts proposed in the methodology are illustrated by examples taken from this work.

The aim of the Microbes project is to identify soil functioning by studying soil biota (microorganisms, fauna and roots) evolution. In this context, a work (presented in [MCD+07]) aims at reproducing earthworms effect on the soil structure and the nutrient availability. Creating a soil model leads to complexity issues. A soil is a multi-scale heterogeneous, three-dimensional and dynamic space. For these reasons, it is difficult to establish a realistic and computable model of soil. RAFALE-SP methodology permits to develop a model and a simulator based on MAS and Fractals: (i) Fractal theory (often used in soil sciences) is chosen to describe the soil and to determine a dynamic environment where agents move; (ii) agents model earthworms.

4. Modélisation Intra-urbaine des Rythmes Quotidiens

4. Meta-models and formalisms to create the mobility model

The aim of this section is to describe more accurately the modelling process of spatialized complex systems in which mobiles move on a area. Describing this type of systems implies to model:

– the space into the MAS environment,

– the individuals into RAFALE-SP mobiles. The mobiles are composed of an agent (the individual's behavior) and a body that represents the physics into the MAS environment (concret representation of the individual),

– mobile skills into perception and motion vector.

Those elements are described into sub-models that describe different aspects of the studied complex systems (mobile mind structures, mobile behavior rules, the space, and so on). We have separated mobile mind that represents the cognition and mobile body that is the concrete representation of the mobile such as suggested in [Mag96, OPFB02]. In this section, these sub-models are presented as followed: (i) At first, the process of modelling the mobile mind is described, (ii) then environments models are pinpointed and finally, mobile skill models, in another words, representation of interactions between agents and their environment are identified.

4.1. *Agent description*

Two models are needed to describe the agents that represent the mind of studied mobiles. The first one identifies the cognitive architecture of individuals. The second one allows the drawing up of agent behavior rules.

4.1.1. *Architecture description*

As said before, mobiles are characterized by a knowledge, skills, goals and behavior rules. The aim of the main architecture description is to give a high level and understandable model that determine knowledge, goals, perceptions and motions that a mobile can have. Note that the mobile body is modelled into the environment description.

UML Class Diagrams are an interesting tool to describe components of agent (mobile mind). UML is indeed famous in majors research areas as geography or biology where it is used to structure real data. So we take advantage of UML (its simplicity, its readability and its expressiveness) to establish a meta-model which aims at structuring mobile mind. This meta-model is composed of five parts (see figure 2):

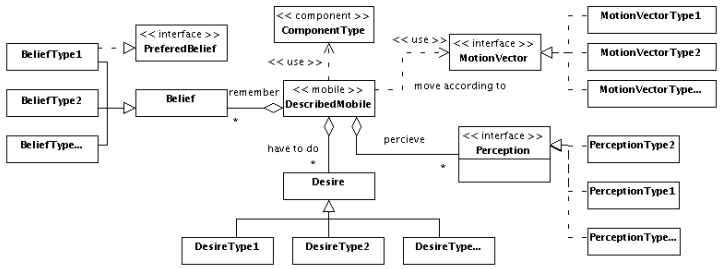

Figure 2: Architecture of agent mind

– *a belief representation* that describes how external elements (streets, other mobiles) are learnt by the mobile.

– *a goal representation* (desire) that defines the objectives of the mobile.

– *a perception of the environment* that describes how a mobile sees other elements located on the environment.

– *a physical capacity* (motionVector) that describes how a mobile evolves on the environment.

– *a behaviour* (described mobile) that describes a motion strategy.

This UML meta-model does not however provides a support to establish a specific model that represents knowledge, goals and abilities of individuals of a real complex system. For this reason, behaviors are determined in Ploom-Unity specification, a specification formalism that we created to express motion in agent based systems. The language is defined further in part 4.1.2. To illustrate this mobile description, a model of pedestrian is presented on the figure 3

This model characterizes a pedestrian that is able to remember services located on *Buildings*, and the map of the town (*Way*). This virtual

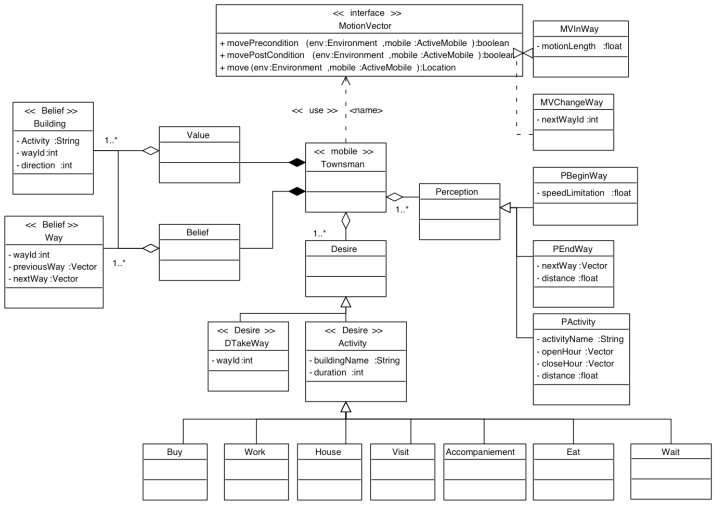

Figure 3: The case of Miro: agent mind architecture of townsman

pedestrian can do motion actions to move into a road (*MVINWay*) and to move from a way to another one (*MVChangeWay*). According to its location in the virtual world, an agent is able to perceive services (*PActivity*) and streets (*PBeginWay, PEndWay*).

Note that, this model can easily be used and understood by scientists coming from various research areas as it is based on UML. It can also be used as support for a computer scientist to collaborate with other researchers coming from other domains.

4.1.2. *Behavior specification*

The aim of the Ploom-Unity specification is to describe with accuracy the mobile mind and its functioning. This specification is in the continuity of the architecture description. It extends UML class diagram and outlines the behavior rules of the mobiles.

A Ploom Unity specification is composed of 8 parts, as illustrated on the figure 4.

– the *head part (clauses "Extends" and "Refines")* defines the identity of a mobile class. It gives the name of the parent class and includes

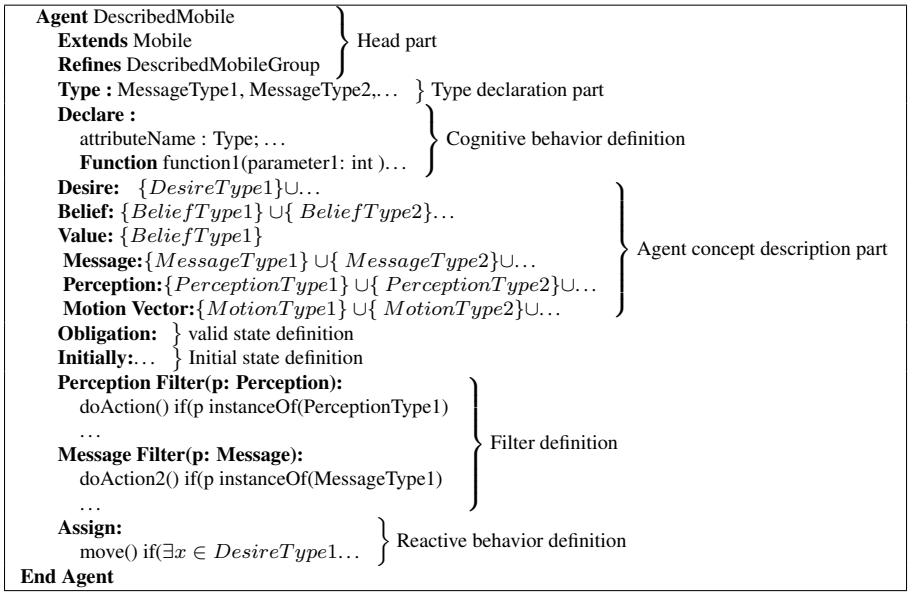

Figure 4: An extract of a specification

mobiles in groups through the `Extends` and `Refines` clauses.

– the *type declaration part (clause "Type")* describes categories of objects that are used in the specification.

– the *cognitive behavior definition (clause "Declare")* shows attributes that are employed by agent methods. These functions compose the behavior of a mobile as they are called according to the mobile state.

– the *Agent concept description part (clauses "Belief", "Value", "Desire", "Message", "Perception" and "Motion Vector")* defines agent knowledge and agent desire. Indeed, it shows the elements that can be saved by a mobile category.

– the *valid state definition (clause "Obligation")* expresses a valid mental state by a predicate.

– the *initially part (clause "Initially")* intends to define the state of the mobile at the beginning.

– the *filter definition (clauses "Perception Filter" and "Message Filter")* determines how the mobile reacts to an event coming from the

environment or another agent.

– the *reactive behavior description (clauses "Assign")*, the kernel of a mobile, expresses the actions be to performed when an event occurs. This section of the specification is composed of instructions that call the corresponding function when the associated condition is true.

From this specification and from the UML agent description, the specification presented on figure 5 has been established. This figure presents in detail behavior rules of pedestrians in the case of the Miro Project.

```
Agent townMan
   Type : shortestPath
   Declare :
      isEndWay : boolean;
      pedestrianSpeed : float;
      ...
      goInWay(p : Perception) : void
            ...
         End Function
      goOutWay(p : Perception) : void
         Begin Function
            ...
         End Function
   Desire:  DActivity, DTakeWay
   Belief: Way, Building
   Value: Building
   Perception: PBeginWay, PEndWay, PActivity
   Motion vector: MVInWay, MVChangeWay
   Initially:
      pedestrianSpeed=1.38888 //default walking speed in cell per second
   Perception Filter(p : Perception):
      goInWay(p) if(p instanceOf(PDebutPerception))|
      ...
      ...
   Message Filter(m : Message):
      ...
   Assign(firstDesire : Desire):
      move(MVInWay(movingSpeed)) if(isArrived=false&&isEndWay=false)|
      ...
End Agent
```

Figure 5: the case of Miro: extract of Townsmans' specification

When this specification is done, agent knowledge, goals, skills and behaviors are determined. Created models are accurate enough to allow the implementation of agents (mobile mind) in a simulator. But before, mobile body and mobile environment must be modelled.

4.2. *Environment description*

The environment models the space where studied mobiles evolve. It represents, for example, a town with its roads, buildings, shops and so on. In addition, it completes mobile descriptions by determining the body of simulated mobiles.

One of the benefit of the RAFALE-SP methodology is that it allows the description of active environments which evolve during a simulation and affect studied mobiles, their actions and their motions. A simple behavior can be defined for each component of the environment (shop, signals) and it allows the reproduction of the global dynamic of the complex system (e.g. traffic of a town). This strategy permits to describe studied mobilities at a microscopic level with agents while other dynamics may be observed at a macroscopic level through the environment. It is important to take into account dynamics that impact on the studied mobilities in order to obtain a realistic model. But, it is not necessary to give a fine grain description of them.

Environment modeling process is closed to the individual modelling scheme presented in section 4.1. The environment modelling process starts with an architecture description and finishes with an environment specification.

4.2.1. *Architecture description*

The RAFALE-SP framework is generic enough to design and simulate various kinds of spatial environments. So, as for agents, the environment model is defined through its characteristics, mainly its architecture, and its behavior proposes a generic environment architecture that scientists (user of the RAFALES-SP methodology) must extend to generate their own mobility model and their mobility simulator. The environment structure is composed of the following features:

– the **space**. It describes the structure of a space by dividing evolution world into atomic parts. For example, we can model a real space like a town by a regular grid. A graph is another solution where nodes represent crossroads and edges define streets.

– the **location**. It defines how mobiles are located on the space. It is a generic structure which must be refined to determine a specific reference

system. For a given space architecture, various reference systems can be identified (e.g. for locating mobiles on a regular grid we can either use “x,y” coordinates or a cell number based reference system).

– the **environment objects**. It defines objects and mobiles that have a position on the environment. We can enumerate two types of located elements. An *ActiveMobile* is a mobile we study in the space. This element is associated with an agent. So its state and its location may change according to the agent behavior rules. *PassiveObjects* are managed by the environment. They represent objects like buildings or shops. Their state is modified by the environment process according to a predefined algorithm.

Figure 6 gives the meta-model of the environment architecture. This meta-model should be extended to be applied to a specific case study. For the Miro project, the model presented by the figure 7 has been established.

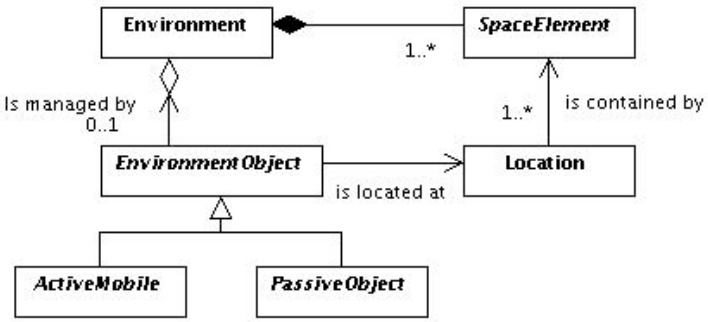

Figure 6: Architecture of the environment

Note that depending on the space structure and on the chosen reference system, observed mobilities are different. A movement results in a location change that has to be treated precisely [Mar05]. It can be represented by a vector that specifies the location transformation. Consequently, an observed mobility depends on the chosen space and reference system. So the choice of a space and a reference system depends on the mobility that the scientist wants to study.

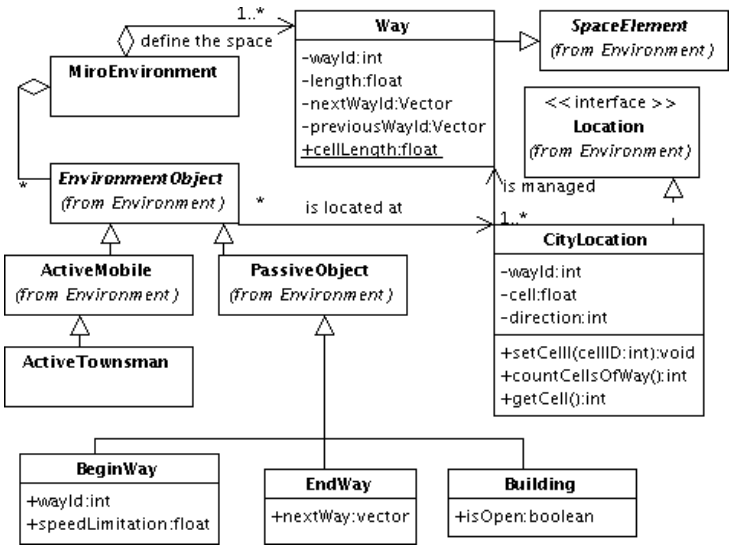

Figure 7: the case of Miro: architecture of the environment

In the case of the Miro project, mobiles are located on the virtual map thanks to the road identifier and to the cell number of the current road. This approach is more accurate than simply locate an agent with cellular automata where parts of different roads may be handled by the same cell.

Space and reference system concepts are one of the keys of the genericity of the RAFALES-SP approach. Thanks to the meta-model generic description of spaces and reference systems, various kinds of movements can be modelled. Thus, various mobilities can be observed. Observed mobilities result in mobile (Agent + ActiveObject) motions. These motions are disturbed by the environment dynamics.

4.2.2. *Behavior specification*

The aim of environment specification is to identify space dynamics. As said before, the dynamics of the environment are expressed by *Passive Objects*. So, the modeling of these dynamics intend to identify behaviors of passive objects.

A passive object is composed of a set of attributes: several variables that determine a state of the passive object. The change of passive object state characterizes the environment dynamic. This dynamic is perceived by mobiles and disturbs them.

To model environment dynamic, an algorithm is associated with every kind of passive objects. The aim of this algorithm is to modify values of state variables contained in the passive objects. At the beginning, the algorithm is determined by a standard UML state-transition diagram (see figure 8). This model gives a synthetic and user-friendly description of the environment behavior. Later, it must be refined by a specification. This second model characterizes with accuracy the behavior of passive objects.

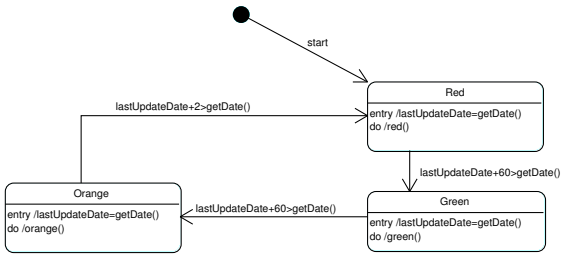

Figure 8: statchart describing PassiveSignal environment object

One specification model is established for each kind of passive object. It declares variables and states respectively identified by the UML class diagram and the UML state transition diagram of the environment. This specification is named by a *header* and is composed of three parts (figure 9):

– *declaration*: enumerates the feature attributes,

– *initial state*: defines the initial state of the passive object,

– *comportment rules*: defines the behavior of the passive object.

Specifying the behavior of the environment allows to associate mathematical or stochastic models with a MAS model. Mathematical or stochastic models are introduced in the MAS environment to represent the surrounding dynamics that are not studied themselves while mo-

```
PassiveObject PassiveSignal
   Declare :
      color : String
      lastUpdateDate : int
      Function red( env : crossroad) : void
         Begin Function
            color=”red”;
            lastUpdateDate=env.getHour();
         End Function
      ...
   Initially:
      color=”red”;
      lastUpdateDate=env.getHour();
   Assign(env : crossroad )
       red(env) if(color=”orange”&& lastUpdateDate+5>env.getHour())|
       orange(env) if(color=”green”&& lastUpdateDate+60>env.getHour())|
       green(env) if(color=”red”&& lastUpdateDate+60>env.getHour())|
End PassiveObject
```

Figure 9: Specification of a crossroad

biles reproduce studied dynamic through agent actions (mobile mind) and active object motion (mobile body).

4.3. *Interaction between agent and environment*

Interactions between a mobile and its environment are based on the agent perception, which is the environment action on the agent, and agent movement, which is the agent action on the environment.

We have adopted a secure approach to manage motions and perceptions of mobiles. To do an action (e.g. a movement), a mobile, in particular its mind (the agent), generates a *Motion vector*. This action is applied on the environment according to predefined transformation rules. These rules modify environment state. For example, they move the mobile on the space (i.e. they change the mobile's body location: the position in the space of the associated active object). Then, the environment computes a new *Perception* for the mobile and send it to the agent.

In the UML class diagram of a mobile's mind, *Motion Vectors* and *Perceptions* are identified. These elements model actions and perceptions that the described mobiles are able to do. Thanks to Ploom-Unity

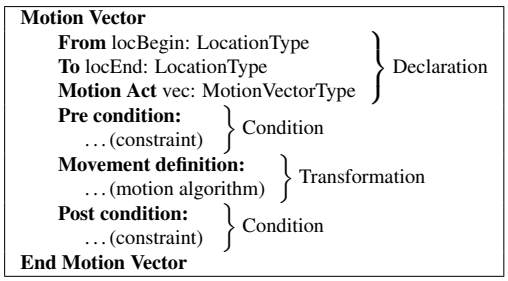

Figure 10: Structure of motion specification

Motion Vector
From locBegin: CityLocation
To locEnd: CityLocation
Motion Act vec: MvInWay
Mobile mob: Townsman
Pre condition: $locBegin.cell \leq locBegin.countCellsOfWay$
Movement definition:
locEnd.wayId=locBegin.wayId
locEnd.direction=locBegin.direction
locEnd.setCell(locBegin.getCell()+vec.motionLength/Way.cellLength)
Post condition: $locEnd.getCell() \leq locEnd.countCellsOfWay()$
End Motion Vector

Figure 11: case of Miro: Moving in a way

specifications, transformation laws are identified. The aim of this section, is to present these specifications (their goal, their architecture and their use).

4.3.1. *Motion vector*

Ploom-Unity is used to describe with accuracy how mobiles move on space. This specification has two main goals: (i) it shows how *MotionVectors* are applied on a space in order to achieve movement, and (ii) it defines when a motion is allowed or not.

A Ploom-Unity specification of Motion Vector contains an algorithm that defines mobile movement and constraints to limit its possible abilities. The specification is divided into three parts (see figure 10):

1) **a declaration part** defines variables which contain the current mobile location (*locBegin*), the future mobile location (*locEnd*) and a

Motion Vector (*vec*) associated with the agent.

2) **condition parts** aim at specifying constraints to limit the mobile's motions. These predicates verify that, from the declared location *locBegin*, a motion can be achieved according to the Motion Vector *vec*. We differentiate two types of constraints. The first one, *Pre condition* confirms that the agent location is correct before a motion. The second one, *Post Condition* verifies that the trip can be done, in other words it validates that the destination location is on a valid part of the space.

3) **a transformation part** performs the future agent location. This part contains the algorithm (transformation rules) that change a location or a state of a mobile into a new one. According to the current location (*locBegin*) and the motion act (*vec*), this part computes the value of the destination location variable (*locEnd*).

Two motion constraints are necessary to confirm movement. The *Pre condition* verifies that an agent is situated at an authorized location before a motion and it allows to verify that a Motion Vector can be applied. The *Post condition* do the same verification but after the action. This constraint is important because it avoids situations in which an agent is located out of the space.

According to the location definition given in figure 11, the specification of *MovingInWay* Motion Vector has been established. This moving action increments (or decrements) the distance of the agent location from the beginning of the way. In this case, the Pre and Post conditions ensures that the agent is not going outside the way.

A motion specification describes with accuracy how an agent modifies the state of the environment. The perception specification shows how the environment modifies agent perception.

4.3.2. *Perception*

UML diagrams show how a mobile perceive elements located into the space. Ploom specifications define algorithms which aim at converting environment objects (passive or active objects) into perceptions understandable by agent (mobile mind).

A perception specification is divided into two parts (see figure 12):

– **declaration part**. It defines a variable that contains an observed located element (*viewedElement*) or an agent perception. This value is computed thanks to the perception algorithm.

– **transformation part**. It performs perception conversion from the state of active or passive object (noted *viewedElement*) to an internal agent perception (*internalPerception*). This part is often a set of affectations but it can also contains complex instructions that model complex phenomena (perception perturbations, troubles and so on).

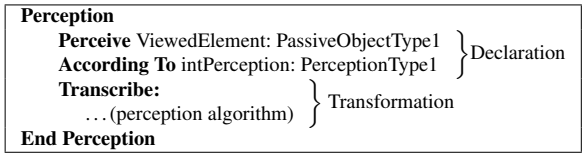

Figure 12: Perception Specification

The specification presented on figure 13 has been developed from the Agent Architecture Class Diagram (figure 3) and the Environment Architecture Class Diagram (figure 7). This specification realizes a simple matching between the attributes of a passive object of an environment and the attributes that compose the perception of the agent. According to the proposed model of the system, this matching could be composed of stochastic rules that model disturbances in perceptions.

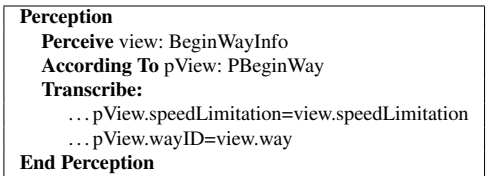

Figure 13: Case of Miro: Perception of a way access

The motion and perception specifications are a progressive process that permits to model an environment characterized by its dynamic and individuals according to an identified behavior. The motion and perception specifications link the individual model and the environment

model. For example, during the architecture description step, perception structures of mobile are identified in an UML class diagram (diagram 3 for instance). These perceptions are a representation (for the agent) of active or passive objects modeled in the environment UML class diagram. Then, constraints are assigned to each perception of mobile (see agent specification on figure 13) in order to reduce mobile's perception. Finally, perception specifications determine transformation laws that convert active or/and passive objects of the environment into perceptions. When the modeling process is finished, the simulator can be implemented.

5. A toolkit to develop simulators

The toolkit supports the final step in the simulator development. It contains specific primitives that developers have to extend and services that manage simulation runs, persistence and distribution. The aim of this section is to describe the architecture and the functionalities of the toolkit. We also present its use in the implementation of two simulators: daily journeys of a city inhabitants and soil biota.

5.1. *Overview of RAFALES-SP toolkit*

The RAFALES-SP toolkit architecture can be considered either from the simulator viewpoint, the simulator architecture, or from its internal organization viewpoint, the toolkit architecture.

5.1.1. *Simulator architecture*

The toolkit contains features which help developpers to implements agents and environments. This framework is based on the *MadKit* [GFM00] multi-agent platform.

MadKit is based on an Agent Group Role (AGR) approach. It contains basic features which manage agent life cycles or agent interactions. It has been developed in JAVA and thus can be used on different operating systems. Several *Madkit* servers, installed on different computers, can work together. For the toolkit implementation, we take

advantage of basic functions of *Madkit* : interaction, life cycle, and distribution management.

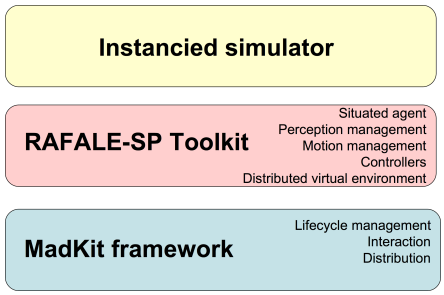

Figure 14: different level of the framework

Our framework extends *Madkit* basic features and contains specific algorithms that be employed by users to implement their own mobiles and their own environments. For example, motion specification is implemented on two abstract functions of environment. The first one contains motion constraints. The second one implements location algorithms. For this reason, simulators developed from the RAFALE-SP toolkit need the Madkit and RAFALE-SP plugins to be run. The development of these simulators is simplified by to primitives of RAFALE-SP dedicated to mobility simulation.

5.1.2. *Toolkit architecture*

The RAFALE-SP toolkit propose a pattern of simulator that scientist have to extend to develop their specific simulator as urban mobility simulator for instance. It is, of course, an oriented agent-based framework and it already contains several classes to develop and manage agents, geographical environment and distribution and so on.

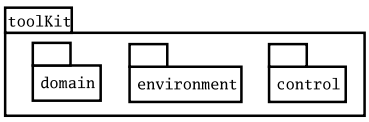

Figure 15: General structure of the toolkit

To support different operating systems, the JAVA language was chosen to develop this framework. In addition, major agent based platforms

and simulators are developed in JAVA so that we can easily interact with these developments.

The toolkit is composed of three packages that correspond to the RAFALES-SP meta-model (see figure 15):

– *Domain* contains every classes we can extend to implement mobiles and to manage them.

– *Environment* is composed of elements which intend to describe a space as a virtual city with streets and locations (building, house, shop).

– *Control* package contains primitives that permit to develop the graphical interface of simulators.

A simulator is the result of a development step, which aims at extending classes of *Domain*, *Environment* and *Control* packages and a compilation step. Generated simulator can be distributed on grids, as we will see in the next parts.

5.2. *How to use RAFALE-SP toolkit to develop simulators*

As said before, the RAFALE-SP Toolkit follows the meta-model structure. Classes and constraints defined in the models are implemented in dedicated classes, functions and variables of the toolkit.

For example, from an agent architecture UML class diagram and an environment class diagram, several motion vectors are identified and modeled by Ploom-Unity specifications. Each Ploom-Unity specification is translated into a specific java class inherited from an abstract java class of the RAFALE-SP toolkit, called *MotionVector*.

The specification presented on the figure 10 shows that a motion vector is characterized by: (i) a pre-condition ensuring that the action can be applied in the current mobile state; (ii) transformation rules that modify a mobile position and are modelled by a specification; (iii) a post-condition ensuring the coherency of the mobile state after applying motion. The *MotionVector* JAVA class architecture is closed to the motion vector specification. It contains 3 main functions to be coded in JAVA: pre-conditions (*canMovePre*, see figure 16), transformation

```
protected boolean canMovePre( EnvObject en, MotionVector mv )
{
   //Select the type of the motion
   if ( mv instanceof MVInWay ) {
      //Cast a MotionVector to a MVInWay motion vector
      MVInWay vec = (MVInWAY) mv;
      //verifies the pre condition of the motion specification
      if(((CityLocation)en.getLocation()).getCell()>=((CityLocation)en.getLocation()).countCellOfWay())
         return false;
      return true;
   }
}
```

Figure 16: An extract of motion check function

```
protected void  moveMobileAccordingTo( envObject en, MotionVector mv )
{
      MVInWay mouvement = (MVInWay) mv;
      CityLocation loc = (CityLocation) (en.getLocation());
      loc.setCell(loc.getCell()+( motionLength/Way.cellLength );
}
```

Figure 17: An extract of motion function

rules (*moveMobileAccordingTo*, see figure 17) and post conditions (*canMovePost*).

Figures 16 and 17 are code extracts from the MIRO simulator implementation. These functions implement pre-condition and transformation rules defined in changeWay specification.

Note that the approach used to develop simulators is closed to motion vector implementation. RAFALE-SP toolkit can be viewed as a simulator canvas that is extended by users and configured to develop their mobility simulators. Moreover, features to translate automatically Ploom-Unity scripts into JAVA codes and to compile simulators are needed to obtain a complete, user friendly methodology. It is one of our further works.

5.3. *Toolkit services*

Each person that wants to use the framework can take advantage of multiple services that are included in it.

Before speaking about these services, note that a city simulator, created with the presented toolkit, already uses several technologies which come from distributed system area. It is based on a MAS associated with Enterprise Java Beans (EJB) and a database (see figure 18). Agents are organized into four categories:

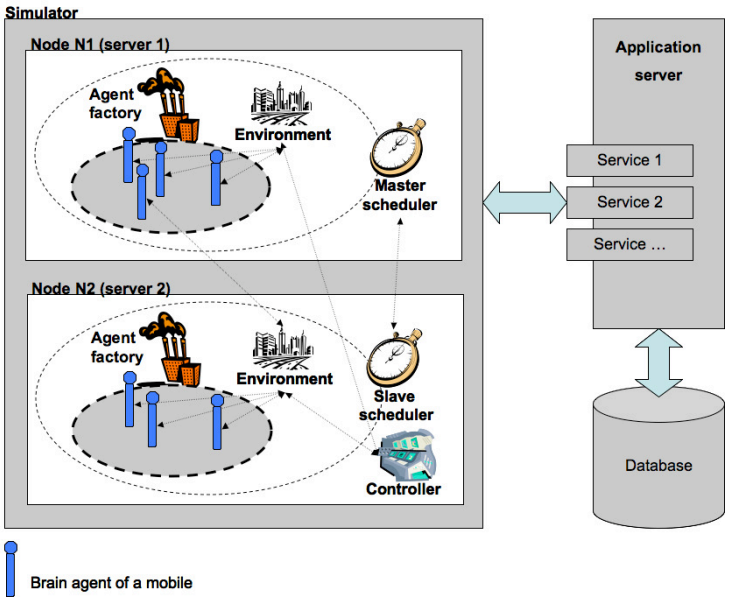

Figure 18: Structure of a simulator

– *environment agents* aim at representing a virtual city. They manage space elements like buildings or roads, and mobile motions.

– *control agents* allow users to interact with a simulation (seeing motions, modifying simulation parameters).

– *mobile agents* represent mobiles we want to study.

– *master agents* manage mobile agents. They can create or destroy them. They are allowed to modify mobile parameters. For example, a master agent can stop an agent evolution when a control agent ask a pause of a simulation.

Beside this quite simple scheme, we had to face some problems : overload from multiplicity of agents, synchronization between agents

when they are distributed on several machines, for instances. Solutions to these problems are presented in the following.

5.3.1. *Management of distributed simulations*

Since our agents are not reactive ones, each of them may be quite heavy in terms of system resource consumption. For this reason, to be able to simulate enough agents to be realistic in a large system, we may need to distribute a simulation on several computing nodes. In this case, the environment must be distributed.

In RAFALE-SP, several computers can work together to execute the same simulator. On each site, there is a master agent and an environment agent. It allows us to divide a virtual space into areas. Each area is managed by an environment agent. The associated master agent takes care of mobiles which move on the area. Obviously, we need to have a pertinent division of the space in order to equitably share the load between computers. We plan to work on an automatic mechanism to handle this kind of problem.

As most of the simulations are synchronous, there executions are based on a common global time. Representing time in a distributed simulator is an important problem that we have solved by implementing a distributed clock. We used an independent clock on each node of the simulation. These clocks are synchronized thanks to message exchanges between the local clock and a master clock. Those messages occur on pre-defined period.

5.3.2. *Data persistence*

We have developed generic classes that easier the connection of a simulator with an application server. These classes use the entity bean of EJB specification to provide access to standard data bases. Their goal is to manage data that can be stored in a database. Thus, information saved in a database are accessible through these objects and there is no SQL code in the agent source code, these generic classes keep them portable.

For example, we used these classes to manage streets, crossroads or locations in MIRO application. Actually, we also plan to create differ-

ent kinds of entity beans that will help, for all the geographical related simulations, to save information into an OpenGIS database.

5.3.3. *Code mobility and remote agent management*

Simulators executed on a network of computers are composed of several simulation nodes (1 simulation node per computer). The simulated agents that represent mobiles are placed on different servers of the network. To enable remote instantiation of simulated agents and their mobility, RAFALE-SP provides a server called *Factory* that is executed on each node of the network. This factory receives mobile creation messages from the user interface and loads simulated agents in the computer memory.

Because of the simulator's distributed architecture agents that represent studied mobiles must exchange messages through the network to cooperate. Moreover the environment may be distributed on several nodes, so agents will have to interact with different parts of it and then with different nodes. To limit remote messages, which cost much resources and time than local ones, simulated agents can be moved from one server to another. The advantage of code mobility principle is that, by transferring an agent from a server to another, it transforms remote messages into local messages and thus it reduces the exchange cost.

The code mobility primitives are implemented in agent factories and these functions can be used by RAFALE-SP developers to manage the agent distribution. The automatic object location is however not taken in charge by the framework. This has to be done in the simulator implementation, by the developer. The framework just offers the support of agent mobility and the developer is free to define the rules of agents location on the different nodes.

5.4. *Implemented simulators*

In section 3.4, we have introduced two examples that illustrate the use of the RAFALE-SP methodology. In this part we present the implementation of the simulator for these examples. We give a global view and details on these implementations and we explain the benefits gained by applying RAFALE-SP.

The MIRO project models and simulates daily journeys of Dijon inhabitants. In this case, the created model describes the town as a graph in which nodes represent crossroads and edges describe roads. Some parameters are associated with each edges and nodes in order to model speed limitations, road signs and so on. On this virtual environment, buildings are located in order to model town services (e.g. houses, firms, restaurants or shops). Dijon inhabitants (mobiles of the model) are modeled by cognitive agents. These agents can learn the town map, service locations and transport durations. They move according to a set of tasks they have to do. The Miro model is implemented on a distributed simulator which have been tested on a network composed of four computers. This simulator is based on the RAFALE-SP toolkit (the kernel of the simulator), the JoNAS [5] application server (to store the town map) and Sicstus prolog to compute the timetables that organize the daily activities of the agents. At the beginning of a simulation, several mobiles are loaded and configured with a set of beliefs and a location. From their knowledge, agents compute a timetable in which the mobile's tasks are defined for one day. Then, mobiles try to accomplish their planning during the first simulated day. They move into the virtual town and modify their knowledge according to their perception. At the end of the first simulated day, agents compute a new timetable which takes into account their past. These new timetables are executed by the agents during the second simulated day and this is reproduced along the different days. After the simulation, results are post-processed by geographers in order to generate maps. From these maps, they extract (for example): (i) space organization of individuals which allow the identification of a major activity for each zone of the space, or (ii) use street traffic rates to determine where traffic jams may appear. The final aim of this work is to make a tool that helps administrations in taking decisions for their urban policy (creating roads, creating industrial zone and so on).

The MIRO project was the first project where we applied the RAFALE-SP methodology. So giving a feedback on the use of the methodology in this project is not easy as we did not have much background on it. It has however been a good validation case due to the size

5. http://jonas.objectweb.org

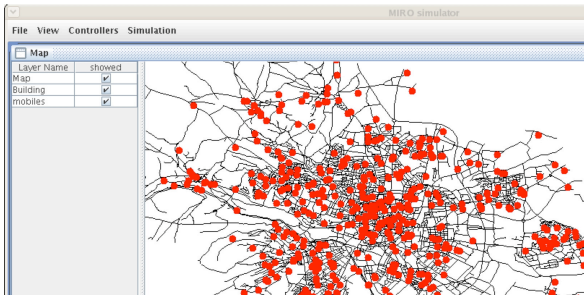

Figure 19: Screenshot of the Miro simulator

of the project. Some parts of the project even lead to improvements in the methodology. So the main benefit that we get from the use of the methodology in this project is the genericity of the designed components and the ability to reuse them in other simulators.

The Microbes project was the second development we made with RAFALE-SP. From the Microbes project, we can state the interest of the methodology. The development phase has been shortened and the model enables to give a realistic virtual environment in which soil concentrations are kept. Simulations give indeed realistic results: the architecture of the virtual soil obtained after a simulation are similar to real soils observed on the ground. There are many mathematical and statistical models that study water retention [WGA08, BPR01], erosion of soils [GCB94, LHT+08]... But there are only few models that focus on internal soil functioning and internal soil dynamics. One of these individual based models (IBM) focus on microbial dynamic (MIOR) [MCB+07]. A second one focus on earthworm impact on sediment [CFCB02]. But, according to our knowledge no one reproduce earthworms' dynamics and their impact on soil structure. The sworm model and IBM are a new way in soil sciences to obtain data that cannot be obtained by experiments, to well understand soil functioning and determine suitable pastoral politics.

The experiences of the MIRO and the Microbes projects show that RAFALE-SP can be applied in a large span of domains to study various complex systems. The key of this ability is given by the genericity of the environment definition and the mobile descriptions. Enabling to de-

termine the "location" is the key feature that allows modelers to define various kind of 2D or 3D virtual environments (continuous, grid, fractal based or GIS based environment). Enabling to define internal minds as complex as users want (with knowledge or not, with perception or not, ...) is the key feature to describe various kinds of mobiles (Human, insects, robots, ...).

In spite of this genericity, users are well guided by RAFALE-SP to establish the model of a complex system and to implement it into a simulator. UML meta-models and Ploom-Unity specifications "determine" a minimal architecture of the mobility-model. To create RAFALE-SP compliant model, scientists have to ask themselves important questions about the complex system and to give answers. As a consequence, using RAFALE-SP, as for every methodology, contributes to a better understanding of the studied system.

The major lack of RAFALE-SP approach (methodology and tools) is the absence of user friendly automation to create models, to translate models into new ones or into runnable simulators. This lack forbids scientists to use the approach without the help of a computer scientists.

As for every methodology, using RAFALE-SP implies to acquire some expertise on the concepts, their implementation, etc. before starting a project. This time is however balanced by the quality of the final realization, the evolution possibilities and it will just impact on the first development as it will be reused in latter developments.

6. Conclusion

In this paper we have presented the RAFALE-SP methodology. This methodology guides scientists in the development of mobility simulators from the early design to the realization. As the agent paradigm provides a good genericity, it has been used in the methodology to characterize mobility. The domain of agent modeling is widely explored, so we have coupled two modeling approaches in order to cover the design and development processes from the beginning, the specification, to the end, the implementation. The first approach defines the general modeling-simulation process and the second one splits up a multi-agent

system in elementary entities: agents, environment, interactions and groups. The methodology is composed of four steps: structural analysis, behaviors analysis, specification and implementation. The first steps are based on a UML meta model and a specification. The last step uses a simulation library to implement the final simulator.

The originality of this approach is that, on the one hand, we associate two well-known modeling approaches to propose a global methodology and, on the other hand, we only model mobiles that we want to observe. Each observed mobile is represented by a cognitive agent and mobiles with simple behaviors, as stochastic behaviors, are defined in the environment.

The RAFALE-SP methodology is fully functional and it has been tested on several simulation projects as the study of individual movements in a city or the study of earthworms in the ground. There is however a limitation on the group notion that is represented in the UML metamodel and in the Ploom-Unity language. This notion has not been implemented at the simulation step as we did not need it in the two projects, but no technical difficulty prevents to implement it and it could be an interesting notion to simulate structured communities.

In the future, the methodology will surely evolve to gain in genericity, flexibility and comfort. The genericity and flexibility will be gained through the use of the methodology to simulate various contexts and by adding new functionalities to fulfill specific needs. The comfort will be improved by developing tools to support specification and implementation processes. Currently, the methodology steps are done one after another by the developer but it is possible to provide semi-automatic or automatic tools to go from one model to another. For example, from the specification level, we could automatically generates skeletons of the main classes to facilitate the implementation.

To improve the methodology, we are currently working on the verification of the models. To achieve this, we need to apply verification rules on our models, either on UML schemes or on an action language. This last can be Ploom-Unity or another one that may better fit to verification rules. Thus, we currently assess and compare Ploom-Unity

(slightly modified) and action calculus that can be a solution to represent agent dynamicity.

References

[BBCM00] Christoph Baumer, Markus Breugst, Sang Choy, and Thomas Magedanz. Grasshopper - an universal agent platform based on omg masif and fipa standards. Rapport technique, IKV++, 2000.

[BCL+05] Arnaud Banos, Sonia Chardonnel, Christophe Lang, Nicolas Marilleau, and Thomas Thevenin. Simulating the swarming city: a mas approach. In *The 9th Int. Conf. on Computers in Urban Planning and Urban Management, CUPUM 2005*, London, UK, 2005.

[BG92] J. L. Beckers, R.and Deneubourg and S. Goss. Trails and u-turns in the selection of the shortest path by the ant lasius niger. *Theoretical Biology*, 159:397–415, 1992.

[BH02] Gordon Beavers and Henry Hexmoor. In search of simple and responsible agents. In *The GSFC Workshop On Radical Agents*, pages 257–268, McLean, VA, USA, 2002.

[BLPM02] François Bousquet, Christophe Le Page, and Jean Pierre Müller. Modélisation et simulation multi-agents. In *deuxièmes assises du GDRI3*, Nancy, France, 2002.

[BMO01] Bernhard Bauer, Jörg P. Müller, and James Odell. Agent uml: A formalism for specifying multiagent interaction. In *AOSE*, pages 91–103, Berlin, Allemagne, 2001. Springer.

[BPG+04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. TROPOS: An agent-oriented software development methodology. *AAMAS*, 8(3):203–236, 2004.

[BPR01] N Bird, E. Perrier, and M. Rieu. The water retention curve for a model of soil structure with pore and solid fractal distributions. *European Journal of Soil Science EJSS*, 55(1):55–65, 2001.

[CBF03] C. Carabelea, O. Boissier, and A. Florea. Autonomy in multi-agent systems: A classification attempt. In *Agents and Computational Autonomy, 2nd International Joint Conference on Autonomous Agents and Multi-agent Systems*, volume 2969 of *Lecture Notes in Computer Science*, pages 103–113, Melbourne, Australia, 2003. Springer.

[CFCB02] J. Choia, F. Francois-Carcaillet, and B.P. Boudreaua. Lattice-automaton bioturbation simulator (labs): implementation for small deposit feeders. *Computers & Geosciences*, 28:213–222, 2002.

[CTC05] Radovan Cervenka, Ivan Trencansky, and Monique Calisti. Modeling social aspects of multiagent systems: the aml approach. In *AOSE*, Ultrecht, Netherland, 2005.

[Dav96] Kinny David. A methodology and modelling technique for systems of BDI agents. In *MAAMAW*, Eindhoven, The Netherlands, 1996.

[Dem03] Yves Demazeau. Créativité emergente centrée utilisateur. In *11èmes Journées Francophones sur les Systèmes Multi-Agents*, pages 31–36, Hammamet, 2003. Hermès.

[DLV02] J. De Lara and H. Vangheluwe. Atom3 : A tool for multi-formalism modelling and meta-modelling. *LNCS 2306*, pages 174–188, 2002.

[Fer90] Jacques Ferber. Conception et programmation par objets. Hermes, 1990.

[Fou05] Sébastien Fournier. *Intégration de la dimension spatiale au sein d'un modèle multi-agents à base de rôles pour la*

simulation : Application à la navigation maritime. PhD thesis, Université de Rennes, France, 2005.

[GCB94] M. Goulard, J. Chadoeuf, and P. Bertuzzi. Random boolean functions - nonparametric - estimation of the intensity-application to soil surface roughness. *Statistics*, 25, 1994.

[GCP03] Roman Gruia-Catalin and Jamie Payton. Mobile unity schemas for agent coordination. In Egon Börger, Angelo Gargantini, and Elvinia Riccobene, editors, *10th International Workshop Abstract State Machines, Advances in Theory and Practice, ASM 2003*, volume 2589, pages 126–150, Taormina, Italy, 2003. Springer.

[GFM00] Olivier Gutknecht, Jacques Ferber, and Fabien Michel. Madkit: une architecture de plate-forme multi-agents générique. Rapport de recherche 00061, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, France, 2000.

[JP97] M. R. Jean and S. Pesty. Emergence et sma. In *Intelligence Artificielle et Système Multi-agents, JFI-ADSMA'97*, pages 323–342, La Colle-sur-Loup, France, 1997. Hermès.

[LCRPS04] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan. Mason: A new multi-agent simulation toolkit. In *The 2004 SwarmFest Workshop*, Michigan USA, 2004.

[LHT+08] G. Leitingera, P. Höllerc, E. Tasserb, J. Walded, and U. Tappeiner. Development and validation of a spatial snow-glide model. *Ecological Modelling*, 211(3-4):363–374, 2008.

[Mag96] Laurent Magnin. *Modélisation et simulation de l'environnement dans les systèmes multi-agent : application aux robots footballeurs*. PhD thesis, Université de Paris 6, Paris, France, 1996.

[Mar05] Nicolas Marilleau. An agent based meta-model for urban mobility modeling. In *DFMA*, pages 168–175, Besançon, France, 2005.

[MCB+07] D. Masse, C. Cambier, A. Brauman, S. Sall, K. Assigbetse, and JL. Chotte. Mior: an individual-based model for simulating the spatial patterns of soil organic matter microbial decomposition. *European journal of soil science*, 58:1127–1135, 2007.

[MCD+07] N. Marilleau, C. Cambier, A. Drogoul, E. Perrier, JL. Chotte, and E. Blanchart. Multiscale mas modeling to simulate complex systems: A case study in soil science. In *MACS2007, Multi-agents for modelling complex systems*, Dresden, Allemagne, 2007.

[ML09] F. Meneguzzi and M. Luck. Norm-based behaviour modification in bdi agents. In *Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 177–184, Budapest, Hungary, 2009.

[NNV06] Michael J. North, T. Collier Nicholson, and Jerry R. Vos. Experiences creating three implementations of the repast agent modeling toolkit. In *ACM Transactions on Modeling and Computer Simulation*, volume 16, pages 1–25. ACM, 2006.

[OBCdRC05] Fabio Y. Okuyama, Rafael H. Bordini, and Antônio Carlos da Rocha Costa. Elms: En environment description language for multi-agents simulations. In *the First International Workshop on Environments for Multiagent Systems (E4MAS)*, volume 3374 of *Lecture Notes in Artificial Intelligence*, pages 91–108, Berlin, 2005. Springer-Verlag.

[OMG02] OMG. Meta object Facilitors (MOF). Technical report, OMG, 2002.

[OPFB02] James Odell, H. Van Dyke Parunak, Mitchell Fleischer, and Sven Brueckner. Modeling agents and their environment. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *AOSE*, volume 2585 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2002.

[PBG03] Gauthier Picard, Carole Bernon, and Marie-Pierre Gleizes. Cooperative agent model within adelfe framework: An application to a timetabling problem. In *AAMAS*, volume 3, pages 1506–1507, New York,USA, 2003.

[TDZ08] J-P. Treuil, A. Drogoul, and J-D. Zucker. *Modélisation et Simulation à base d'agents: Approches particulaires, modèles à base d'agents, de la mise en pratique aux questions théorique*. Dunod, 2008.

[WGA08] Y. Wang, S.M. Grove, and M.G. Anderson. A physical-chemical model for the static water retention characteristic of unsaturated porous media. *Advances in Water Resources*, 2008. In press.

[WJK00] Micheal Wooldbridge, Nicholas Jennings, and David Kinny. The GAIA methodology for agent-oriented analysis and design. In *AAMAS*, pages 285–312, Netherland, 2000.

[ZPK00] B. Zeigler, H. Praehofer, and T. G. Kim. *Theory of Modelling and Simulation : Integrating Discrete Event and Continuous Complexe Dynamic Systems*. Academic Press, second edition, 2000.

Marilleau Nicolas, Lang C., Chatonnay P., Philippe L. (2012)

RAFALE-SP : a methodology to design and simulate geographical mobility

Studia Informatica Universalis, 10 (1), 38-76