Biologically Inspired Agents: A Framework for Formal Modelling and Simulation of Agents with Elementary Spatial Attributes

Isidora Petreska

South East European Research Centre at Thessaloniki
Department of Computer Science

The University of Sheffield

A thesis submitted for the degree of

Master of Philosophy

December 2013

To all the people that supported me

and made this happen:

to my mother, Branka,

to my father, Petre,

to my partner, Kostas,

to my supervisors, Petros and Marian,

to my examiners, Dimitris and Ilias.

Abstract

Applications of biological or biologically inspired multi-agent systems often assume a certain level of reliability and robustness, which is not always easy to be achieved. Formal modelling and verification of these systems may present many interesting challenges. For instance, formal verification may be cumbersome or even impossible to be applied on models with increased complexity. On the other hand, the behaviour of bio multi-agent systems consists of communities evolving in space and time (such as social insects, tissues, colonies of bacteria, etc.) which are characterised with a highly dynamic structure. Formal modelling of such systems cannot be carried out in a neat and effective way.

This work presents many interesting problems in the area of modelling and verification of bio multi-agent systems. Targeting a rather broad scope, the path for devising a global solution to tackle all of the problems, can be considered as secularly optimistic. Instead, we discuss a number of improvements in the development process, including enhancements on several modelling formalisms, alternative ways to formal verification, as well as a research framework which changes the standard modelling and verification approach of bio multi-agent systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An *agent-based model* (ABM) is often used to describe complex phenomena as dynamic systems of interacting agents. The ABM agents are often familiar to actions such as adaptation and reproduction, introducing new computational paradigms for modelling these behaviours, which are principally found in biologically inspired systems. Over the last years, formal modelling is considered as one of the most essential stages in software engineering [22] and therefore in the development of *multi-agent systems* (MAS) as well. It is often followed by formal *verification*, a process which employs formal methods to confirm that a model satisfies certain properties [32, 45, 47]. There are varieties of formal methods in agent-oriented engineering (Z [85], VDM [41], FSM [31], Petri Nets [79], and others), and a number of approaches towards modelling and verification biological phenomena. However, as the complexity of a MAS increases, considerable difficulties get introduced in the process of formal modelling and verification. Such large-scale communicating and/or emergent systems are hard to be formally modelled due to lack of expressiveness of the current formal notations for biological and biology-inspired MAS.

Definition 1.1. *Biological* MAS are systems that mimic the behaviour of their biological counterparts. Examples of such systems include insect colonies of ants or bees, flocks of birds, herd movement, cell tissues etc. On the other hand, *biology-inspired* MAS are systems inspired from biological processes. For instance, a flock of model helicopters is a biology-inspired MAS inspired by the flocking behaviour of birds. Both types of systems can be referred to as *bio-MAS*.

Defined as complex systems, bio-MAS should be characterised with reliability, quality and robustness. Therefore, besides modelling, verification and testing could be considered the next important steps in the developing cycle. However, it is fairly straightforward (although not easy) to apply the known verification and validation techniques to biological systems composed from one agent, but *it is extremely hard* to transfer such techniques to MAS. This is particularly the case when the MAS has a dynamic behaviour, i.e. the number of agents and their communication network constantly changes throughout their lifetime. One reason might be the observation of the amount of errors, which increase with the amount of interacting components. Due to the computational complexity, formal verification of a complete formal model, as well as complete testing, *are very hard to be achieved*. Moreover, *it may be also impractical* to apply known formal techniques due to the vast amount of time (combinatorial explosion of state space) and effort spent.

Bio-MAS represent a network of interactions and information flow. Complex systems experience dynamics in a non-linear group level [10, 53, 16]. Thus, even the small changes within the individual agent rules might cause a huge difference in behaviour of the system as a whole, i.e. *emergence*. Some research refers to this relationship (between the individual agents and the system as a whole) at a *micro-level* and *macro-level* and they focus towards investigating the links between them [2]. In agents that operate in a 2 or 3-dimensional space, emergence is characterised by a pattern appearing in the agents configuration at some instance during the operation of the system. Common examples are colonies of social insects, like ants, birds, fish etc. The type of emergence observed is related to the agent's positioning in space, for example formation of a line, flocks, schools, herds etc.

Verification techniques, such as *model checking* are applied to check whether a model satisfies certain properties, whereas *validation* is applied to confirm that a model satisfies the user requirements. Verification and validation of the emergent behaviour of MAS is an extremely complex task. It is not only due to the fact that the verification process leads to combinatorial explosion, but also the fact that emergent properties should be identified first before there is an attempt to be verified. The latter is not always straightforward. It is therefore desirable to combine several formal with informal techniques that would be able to contribute towards the verification of MAS. Moreover, someone could apply formal verification techniques (model checking) under the assumption that a desired emergent property is known, which might not be always the case. Finally, modelling these agents would require modelling of their position; and verification would require the exploration of the state space from the combination of all agent positions evolved through time.

These concepts and properties of bio-MAS may indicate another perspective to look into these systems, i.e. as spatial systems. There might be different approaches for modelling and verification spatial phenomena of bio-systems.

1.1 Aims and objectives

Some of the common problems found in complex bio-systems with non-linear dynamic properties, were already introduced. To highlight:

- As the complexity of a MAS increases (an increase in the number of agents and their communication network), considerable difficulties are introduced in the process of formal modelling and verification.
- Large-scale communicating and/or emergent systems are hard to be formally modelled due to the lack of expressiveness of the current formal notations.
- It is very hard to transfer known verification and validation techniques to MAS.
- Due to the computational complexity, formal verification of a complete formal model, as well as complete testing, are very hard to be achieved.
- It may be also impractical to apply known formal techniques due to the vast amount of time (combinatorial explosion of state space) and effort spent.

- Modelling and verification of bio-agents that operate in a 2 or 3-dimensional space, leads to combinatorial explosion.
- Emergent properties should be identified, before there is an attempt to be verified and that is not always straightforward.

The *aim* behind this work is: Definition of an abstract model supporting elementary geometry and development of a methodology to build agent based systems using this concept which will allow simulation and visualization of systems.

This can be further linked with the following *research questions*:

- Do bio-MAS require explicit spatial features in respect to modelling and simulation?
- How can the existing modelling and simulation techniques for bio-MAS be improved such as to facilitate more reliable and robust systems?

The *objectives* include:

O1: Investigate on spatial systems, modelling formalisms for spatial bio-MAS outlining properties and disadvantages of existing modelling formalisms, as well as verification and simulation strategies and how can they be enhanced to better support complex spatial bio-MAS.

O2: Identifying illustrative case studies which are scalable (experiments with different numbers of agents), simple to be modelled and have spatial characteristics.

O3: Linkage with simulation platforms in order to observe emergent behaviour.

O4: Proving the appropriateness of the method through simulations and visualisations.

O5: Devising a framework that will combine all of the steps of developing spatial bio-MAS into a process to improve the standard modelling and verification approach for bio-MAS.

O6: Extending the definition of approaches for modelling bio-MAS with geometrical elements into a coherent model.

O7: Extending existing tools with features coming out of the new definitions.

1.2 Structure

Chapter 2 talks about modelling bio-inspired systems as spatial MAS. The discussion starts with bio-MAS and their spatial behaviour (Section 2.1), followed by modelling bio-inspired systems (Section 2.2). A number of approaches towards modelling MAS related biological phenomena are presented and discussed in details, such as: Finite state machine modelling approaches are presented in Section 2.4, membrane computing modelling approaches in Section 2.5, and a hybrid modelling approach in Section 2.6. This chapter concludes with comparison of these modelling approaches Section 2.7 and a short summary.

Chapter 3 is a rather small chapter for the different tools for modelling bio-inspired systems as spatial MAS. It is connected to the previous topic on different modelling approaches because the tools that are described in this chapter belong to the modelling formalisms discussed. These tools will be also used in the following chapters for demonstrating models of different case studies.

Chapter 4 introduces formal verification, simulation and validation concepts for bio-systems. Sections 4.2 and 4.3 talk about verification and simulation of the introduced modelling approaches.

Chapter 5 provides introduction of several visual simulation platforms such as: NetLogo, Repast and the Flame visualiser (Section 5.1). Comparison of these simulation platforms based on the several criteria (modelling, implementation, validation, verification, etc.) is presented in Section 5.2. This chapter ends with conclusions on the comparison (Section 5.3) and a short summary.

Chapter 6 introduces a framework for modelling and verification of spatial MAS. Section 6.1 presents some requirements towards development of such a framework and underlines its structure. Section 6.2 talks about two different instantiations of this framework. This chapter can be considered as a rather substantial one because it lays foundations for the work of the following chapters.

Chapter 7 focuses on formal modelling of spatial MAS. Sections 7.1 and 7.2 provide definitions for extending two of the already introduced the modelling formalisms, in order to extend them to support spatial properties. The new formalisms are discussed and supported with examples.

Chapter 8 talks about visualisation and simulation of spatial MAS. Sections 8.1 and 8.2 can be considered as continuation of the previous chapter because they discuss visualisation and simulation strategies on the new modelling formalisms introduced. This chapter also presents a tool developed for translation of a system modelled with a finite state machine modelling approach into executable code of a visual simulation platform.

Chapter 9 presents discussion and evaluation of this thesis, underlying the contribution, evaluation and future work.

Appendix A presents a list of the author's publications sorted in both in chronological order and the order of importance.

Appendix B shows the code of the *Aggressor-Defender* case study in a visual simulation platform, namely NetLogo.

Appendix C shows the compiled code for the *Foraging Ant* case study, with the tool presented in Chapter 8.

Appendix D shows the class diagram of the translator component from the tool presented in Chapter 8.

Chapter 2

Modelling bio-inspired systems as spatial MAS

Originating with the Von Neumann machine, an *agent-based model* (ABM) is used to describe complex phenomena as dynamical systems of interacting agents. The ABM agents are often familiar to actions such as adaptation and reproduction. Being characterized with autonomy, local views and decentralisation, the ABM agents occur to manifest complex behaviour and self-organization even when implementing simple individual strategies.

2.1 Bio-MAS and their spatial behaviour

The most widely accepted definition of an *agent* is the one provided by Wooldridge and Jennings [97]:

> **Definition 2.1.** "The term agent is used to denote a hardware or (more usually) software-based computer system that enjoys the following properties:
>
> - *Autonomy*: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
> - *Social ability*: agents interact with other agents (and possibly humans) via some kind of *agent communication language*;

- *Reactivity*: agents perceive their environment and respond in a timely fashion to changes that occur in it;
- *Pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by *taking the initiative*."

Agents can also have other properties, to name a few [97]:

- Mobility: an agent can be moved from one environment to another;
- Adaptivity: an agent can be continuously adapted to its environment;
- Veracity: an agent communicates false information without its knowledge;
- Benevolence: an agent does not have conflicting goals;
- Rationality: an agent acts towards achieving its goals.

Agents can have different characteristics, such as knowledge, beliefs, desires, intentions, obligations, learning ability etc. There are several common agent architectures corresponding to the different types of agents. In a *deliberative architecture* the beliefs of the agents and the environment state are represented as logic formulae. The actions of the agents are defined in terms of deduction rules. *Reactive architecture* has no explicit reasoning and maps the perceptual input of the agents to actions. The behaviours are rules of the type: `if <situation> then <action>`. In *Belief desire intention (BDI) architecture* an agent is defined in terms of what the agent knows for the world or *beliefs*, what the agent likes to achieve or *desires* and its *intentions*. There are also *hybrid architectures*, such as exploiting both deliberative and reactive agent properties.

A *multi-agent system* (MAS) is a collection of agents with interacting capability, situated in an environment. Agents can serve as computational models to represent the vastly dynamic organization of the biological phenomena in nature, thus giving birth to the nature inspired computing or further, *biological artificial systems*. These systems may take over two directions, namely systems that are used in order to represent a model/simulation of a real biological phenomenon, or the ones inspired by biological phenomena – already introduced in Definition 1.1 as *bio-MAS*.

A large subset of systems within the MAS domain, including bio-MAS, can be characterised as *spatial agents*.

> **Definition 2.2.** *Spatial agents* can be defined as collections of agents distributed and moving in n-dimensional space. They have incomplete knowledge of the environment and can change their direction and position through time.

Spatial agents are distributed through a physical space, usually as a local collection of agents (or computational devices) with the following characteristics:

- The distance between the individual agents has a robust impact on their interaction links; and
- The spatial structure of the system very often characterizes the functional goals of its agents.

MAS such as wireless sensor networks and animal (or robot) swarms, are clear examples of how the distance between the agents can affect the overall network topology of agent communication. At the same time, the overall structure of the system affects solving of spatial problems like: reaction to some spatial variant (temperature), destroying an enemy, etc. It is important to underline that not all systems distributed in the space belong in the category of spatial systems. The behaviour of the spatial MAS can be defined and analysed by observing **spatial concepts** such as: *location*, *region*, *neighbourhood*, *communication*, *perception*, *propagation*, etc.

Finally, their spatial relationships directly affect the internal organization of the agents and the interactions between them. For the purpose of simplicity, one-agent system inspired from biology can be considered with the following case study.

> **Case Study 2.1.** ***The foraging ant.*** An ant randomly moves in a 2-D space. If the ant encounters a seed, it picks it up. If the ant picks up a seed, it carries it back to the base and leaves it there. After

leaving a seed to the base, the ant again moves randomly. The ant can carry only one seed per time (see Fig. 2.1).

Similar to this case study, instead of the ant agent there could be a foraging bee (biology MAS) or a cleaning robot (biology-inspired MAS). It can be noted that all these systems clearly fall in the category of spatial systems as well.

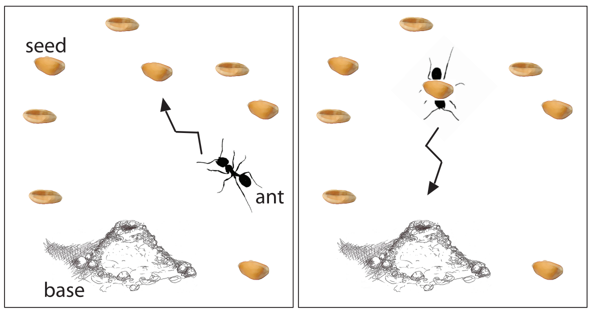

Figure 2.1: Case study 2.1. The foraging ant.

2.2 Modelling bio-MAS

When it comes to developing agent-based systems, there is a software engineering process [96] that describes what is a specification of an agent system, how to implement these specifications and how to verify that the system satisfies its original specifications. This was adapted in the framework for developing bio-MAS (presented in Fig. 2.2) that classifies the development process in four basic steps:

I. Observation of natural phenomena;

II. Modelling;

III. Model implementation or model simulation; and

IV. Verification and testing, where verification can be formal verification (model checking) and informal verification (visualisation or visual simulation).

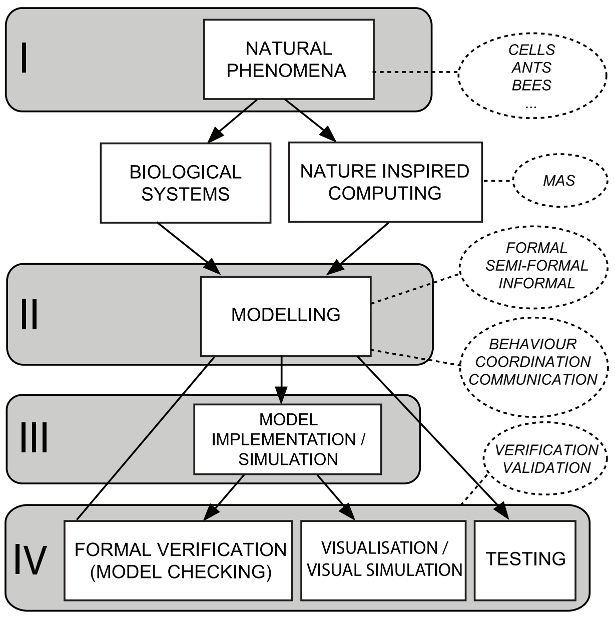

Figure 2.2: Framework for developing bio-MAS.

This chapter has a focus on the first three steps of the development framework (observation of natural phenomena is presented through case studies).

Modelling can be accepted as one of the most essential steps in bio-MAS development and it can be carried out with many different techniques. The step of modelling can be further classified as *formal*, *semi-formal*, or *informal*. Informal modelling refers to usage of visual modelling languages, such as use-case modelling, which lacks a formal definition of their semantic (visual modelling languages can often lead to subjective models). On the other hand, formal specification languages solve these weaknesses and they facilitate creation of models with precise semantics.

As the complexity of a MAS increases, considerable difficulties get introduced in the process of formal modelling. Such large-scale communicating and/or emergent systems are hard to formally model due to the lack of expressiveness of current formal notations. These concepts and properties of bio-MAS gave birth to the idea to *look at bio-MAS as spatial MAS*.

Immediately after the process of modelling, the framework shows the step of *model implementation* or *model simulation*. Model simulation is a representation of the system that can be further divided as:

- Textual model simulation – examples include XMDL [42], JSXM [18], PPSDL [86];
- Visual model simulation (animation) – an application that would animate the execution of ABM such as NetLogo [95, 94].

2.3 Formal modelling approaches for MAS

There are varieties of formal methods in agent-oriented engineering which basically focus on different aspects of the development. Some of them focus on the data structures and operations of a system (Z, VDM [85, 41]), while others into demonstrating the control of its states (FSM, Petri Nets [31, 79]). These approaches are beneficial for modelling systems with a predetermined constant number of agents that is not expected to change. Nevertheless, this is hardly the case in biological systems wherein the simplest model is characterised with dynamic structural changes [88]:

- new agents might be introduced at any point in the lifetime of the system;
- existing agents might discontinue to be a part of the system (i.e. die);
- new communication links between agents might be established;
- existing communication links might be broken; and finally
- agents constantly change their primary identifier (ex. positioning in space or direction) affecting the overall topology of the system.

With the attempt to correctly model a system with these properties, a number of approaches towards modelling MAS related biological phenomena have been developed. Some of these approaches are:

- Finite State Machine approaches:
    - X-machines (XM) and Communicating X-machines (CXM) – specialised into representing the behaviour of biological colonies [42, 46].

- Membrane computing approaches:
    - P Systems (PS) and Population P Systems (PPS) – inspired from the biochemical processes in the living cells that bring in solutions to the dynamic structure of multi-component systems [24, 75].
- Hybrid approaches:
    - The OPERAS framework and its instances OPERAS$_{XC}$ or OPERAS$_{CC}$ – originated as combination of both CXM and PPS targeting the changes occurring in the structure of a dynamic MAS [88].

It is important to highlight:

> *The subsequent work is going to concentrate on these modelling approaches. This decision was based on a survey presented by Beal et al. [3], which analyses and compares a large number of spatial computing domain specific languages.*

According to the survey, the goals of systems in the following domains are often explicitly spatial [3]:

- Amorphous computing;
- Biological modelling and design;
- Agent-based models;
- Wireless sensor networks;
- Pervasive systems;
- Swarm and modular robotics; and
- Parallel and reconfigurable computing.

For this work, the *biological modelling and design* and *agent-based models* domains are of a particular importance. There are many spatial computing domain specific languages throughout these domains identified in [3], including a few additional formalisms that deal with space explicitly. To name a few representative languages [3]:

- Biological modelling and design approaches:
  - Antimony [84], ProMoT [59] and iBioSim [60] allow description of the bio-molecular reactions of cells.
  - P-systems [24] and the Brane calculus [8] are particularly suitable for computations carried-out in biochemical systems of cells and tissues of cells.
  - L-systems [74] and MGS [28] are more explicitly spatial approaches. For instance, L-systems are used to model the growth and structure of plants.
  - Gro [52] is a Python-like language. It was designed for stochastic simulations in a growing colony of Escherichia coli.
- Agent-based modelling approaches:
  - Agent UML [62] and Agent Modeling Language (AML) [90] are graphical agent modelling languages.
  - Jade [51], AGLOBE [11] and The Cognitive Agent Architecture (Cougaar) [30] are agent frameworks.
  - NetLogo [95], Repast [61], MASON [54] and Swarm [58] are agent modelling and simulation toolkits.
- Other modelling approaches:
  - Approaches based on process algebras, such as: π-calculus and Api-calculus (an extension of π-calculus) [57]. This group of formalisms is also suited for modelling systems with dynamic structural changes. Api-calculus addresses knowledge representation, organizational grouping and migration of agents among groups [78].
  - 3π [9] is another extension of π-calculus. It employs the idea of modelling the space as a 3-dimensional geometric space.

2.4 Finite state machine modelling approaches

A XM can be considered as a representative finite state machine (FSM) modelling approach for bio-MAS. It actually resembles a FSM with the power of being more expressive [45]. This property is achieved due to the differences that XM have from FSM, namely they have memory and their transitions have functions that operate on the inputs and the memory values. In this work, the term XM refers to a XM variant particularly defined for modelling purposes, i.e. a deterministic stream X-machine.

2.4.1 X-machines and Communicating X-machines

A deterministic *stream X-machine* is formally described in Definition 2.3. Fig. 2.3 shows an abstract example of a X-machine.

Definition 2.3. A *stream X-machine* is an 8-tuple M = (Σ, Γ, Q, M, Φ, F, q_0, m_0), such that [42, 47]:

- Σ and Γ are input and output sets of symbols,
- Q is a finite set of states,
- M is an n-tuple called memory,
- Φ is a finite set of partial functions that map an input and a memory state to an output and a new memory state,
  ϕ: Σ × M →Γ × M,
- F is a function that determines the next state, given a state and a function from the type Φ, F: Q × Φ →Q, and
- q_0 and m_0 are the initial state and memory respectively.

With the focus on the practical development of communicating systems, a structure knows as *Communicating X-machines* (CXM) can be formed (see Fig. 2.4), providing a way to deal with agents communication [44, 46]. A CXM model consists of several XM models, able to exchange "messages". The term "message" refers to the output of a XM, which can become an input to a function of another XM. In Fig. 2.4, the symbol • denotes that a function receives an input from

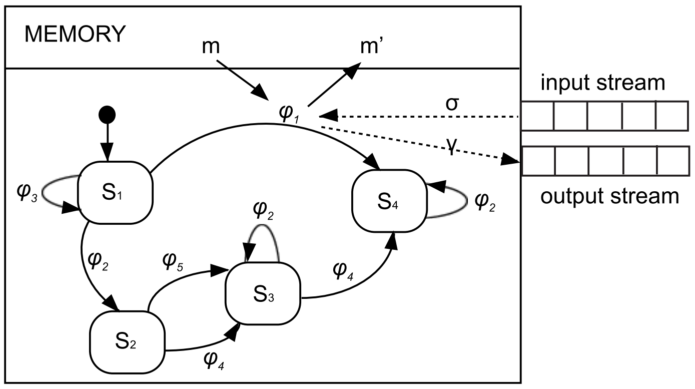

Figure 2.3: An abstract example of a X-machine.

another machine (CX_j), and the symbol ◆ denotes that a XM sends its output to another machine (CX_k).

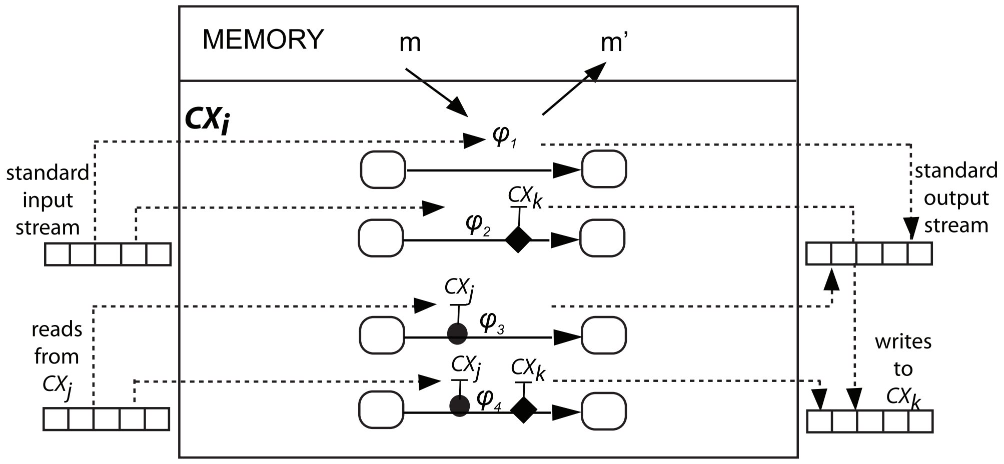

Figure 2.4: An abstract example of a Communicating X-machine.

2.4.2 Spatial agent modelling with XM

Referring to the foraging ant case study from Fig. 2.1, event though it is a very simple example, there might be quite a few different models for it. Fig. 2.5 presents the XM models, from a very abstract to more detailed one using the XM approach.

Table 2.1, Table 2.2 and Table 2.3 demonstrate the three ways of modelling the foraging ant problem. The first solution a) of Fig. 2.5 is presented in Table 2.1.

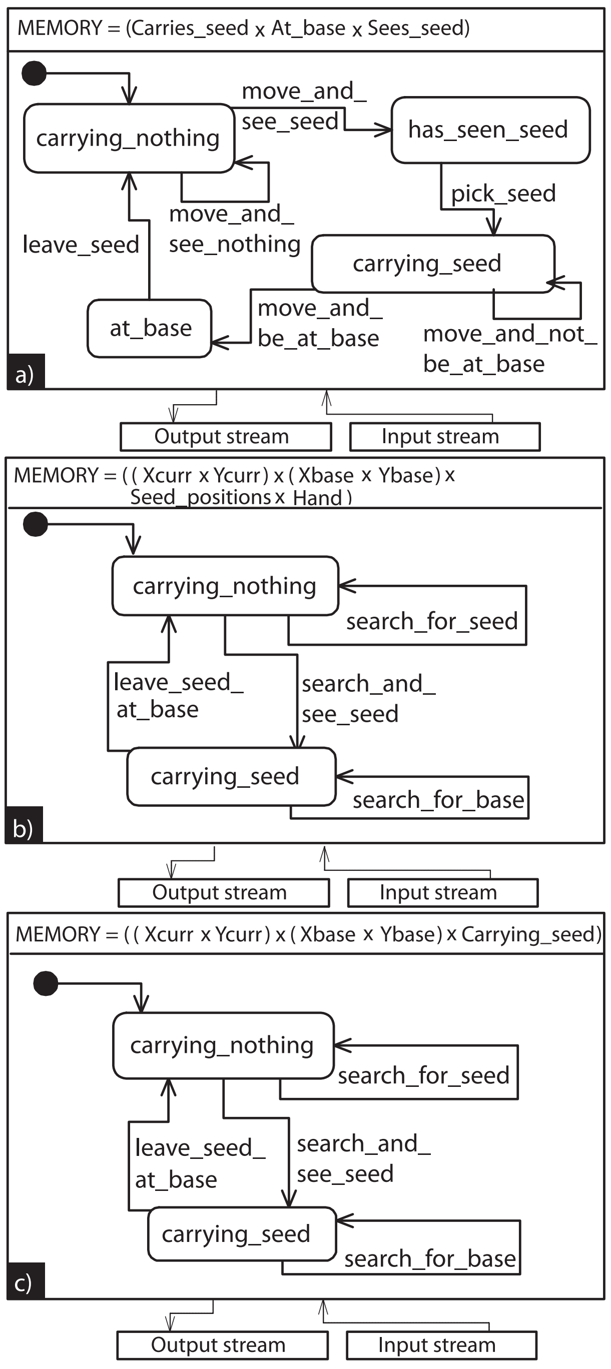

Figure 2.5: Examples of modelling the foraging ant case study: a) "very" abstract representation b) more detailed, but complex representation c) the "best" represented solution

Table 2.1: Foraging ant case study, solution a)

a) Q = {carrying_nothing, has_seen_seed, carrying_seed, at_base}
M = (Carries_seed × At_base × Sees_seed), where Carries_seed, At_base, Sees_seed = {true, false}
m_o = (false, false, false)
q_o = {carrying_nothing}
Σ = {“move to a place w/o seed”, “move to a place with seed”, “pick seed”, “search for base”, “move to base”, “leave seed”}
Γ = {“ant keeps moving empty”, “ant detected seed”, “ant picked seed”, “ant searches for base”, “ant found base”, “ant left seed”}
Φ = {
move_and_see_seed (“move to a place with seed”, (Carries_seed, At_base, Sees_seed)) = (“ant detected seed”,(false, false, true)) if Carries_seed = false ∧ Sees_seed = false,

move_and_see_nothing (“move to a place w/o seed”, (Carries_seed, At_base, Sees_seed)) = (“ant keeps moving empty”, (false, false, false)) if Carries_seed = false ∧ Sees_seed = false,

pick_seed (“pick seed”, (Carries_seed, At_base, Sees_seed)) = (“ant picked seed”, (true, false, false)) if Carries_seed = false ∧ Sees_seed = true,

move_and_be_at_base (“move to base”, (Carries_seed, At_base, Sees_seed)) = (“ant found base”, (true, true, false)) if Carries_seed = true ∧ At_base = false,

move_and_not_be_at_base (“search for base”, (Carries_seed, At_base, Sees_seed)) = (“ant searches for base”, (true, false, false)) if Carries_seed = true ∧ At_base = false,

leave_seed (“leave seed”, (Carries_seed, At_base, Sees_seed)) = (“ant left seed”, (false, true, false)) if Carries_seed = true ∧ At_base = true,
}

This is a “very” abstract representation that does not even take into consideration the position (coordinates) of the ant, or the positions of the seeds. The fact that XM are generic and do not impose modelling of a position, in such an example might result in an incomplete model.

A more detailed representation can be derived from the second solution b) of Fig. 2.5, as seen in Table 2.2. It can be noted that the representation is a design choice, for instance the memory variables that correspond to positions are integers. Yet again, this representation is more complex and probably more difficult for understanding. This is due to the fact that all of the positions of the seeds must be known in advance. Once the ant finds a seed, its position is removed from the set of seed positions.

Finally, the last representation c) of Fig. 2.5 is presented in Table 2.3. This is a better solution with respect to the other two solutions because it is less complex and more complete in terms of the attributes it describes. In this example the seed positions are not known in advance. More comprehensive specification can be found in [63].

This case study demonstrated that there might be different ways to modelling, even for the simplest scenario. The differences in the foraging ant models appear to be in the modelling of the position and the direction of the ant. This leads towards identification of the following shortcomings:

- There might be many different solutions (even for the simplest model) for representing the commonly found properties, such as the initial position or the direction of an agent. This makes it more difficult to read a given model (we have to understand how the modeller decided to represent these properties) and even to create one (every time the modeller has to think how to represent them).
- The memory holds all data structures required, including the position and the direction.

It can be noted that these shortcomings outline the already discussed connection between bio-MAS and spatial systems.

Table 2.2: Foraging ant case study, solution b)

b) Q = {carrying_nothing, carrying_seed}
M = $((X_{curr} \times Y_{curr}) \times (X_{base} \times Y_{base}) \times$ Seed_positions $\times$ Hand)
where X_{curr}, Y_{curr}, X_{seed}, Y_{seed}, X_{base}, Y_{base} = $\mathbb{Z}$, Seed_positions = $\{(X_{seed_1} \times Y_{seed_1}), (X_{seed_2} \times Y_{seed_2}), \dots (X_{seed_n} \times Y_{seed_n})\}$, n $\in \mathbb{N}$,
Hand = {full, empty}
m_o = ((2, 3) , (0, 0), {(2, -3), (4, -6), (2, 1), (3, 5), (-1, 5)}, empty)
q_o = {carrying_nothing}
$\Sigma = (X_{new}\ , Y_{new})$, where $X_{new}\ , Y_{new} = \mathbb{Z}$
Γ = { "ant keeps moving empty", "ant detected and picked seed", "ant searches for base", "ant found base and left seed"}
Φ = {
search_and_see_seed $((X_{new}, Y_{new}), ((X_{curr}, Y_{curr}), (X_{base}, Y_{base}),$ Seed_positions, empty)) = ("ant detected and picked seed", $((X_{new}, Y_{new}), (X_{base}, Y_{base}),$ Seed_positions$\backslash(X_{new}, Y_{new})$, full)) if $(X_{curr}, Y_{curr}) \neq (X_{base}, Y_{base}) \wedge (X_{curr}, Y_{curr}) \neq (X_{new}\ , Y_{new}) \wedge$ Seed_positions $\neq \emptyset \wedge (X_{new}, Y_{new}) \in$ Seed_positions,

search_for_seed $((X_{new}, Y_{new}), ((X_{curr}, Y_{curr}), (X_{base}, Y_{base}),$ Seed_positions, empty)) = ("ant keeps moving empty", $((X_{new}, Y_{new}),$ (X_{base}, Y_{base}), Seed_positions, empty)) if $(X_{curr}, Y_{curr}) \neq (X_{base}, Y_{base})$ $\wedge (X_{curr}, Y_{curr}) \neq (X_{new}, Y_{new}) \wedge (X_{new}, Y_{new}) \notin$ Seed_positions,

search_for_base $((X_{new}, Y_{new}), ((X_{curr}, Y_{curr})\ , (X_{base}, Y_{base})\ ,$ Seed_positions, full)) = ("ant searches for base", $((X_{new}\ , Y_{new})\ ,$ (X_{base}, Y_{base}) , Seed_positions, full)) if $(X_{curr}, Y_{curr}) \neq (X_{base}, Y_{base}) \wedge$ $(X_{curr}, Y_{curr}) \neq (X_{new}\ , Y_{new})$,

leave_seed_at_base $((X_{new}, Y_{new}), ((X_{curr}, Y_{curr}), (X_{base}, Y_{base}),$ Seed_positions, full)) = ("ant found base and left seed", $((X_{new}\ , Y_{new})$, (X_{base}, Y_{base}) , Seed_positions, empty)) if $(X_{curr}, Y_{curr}) = (X_{base}, Y_{base}) \wedge (X_{curr}, Y_{curr}) \neq (X_{new}\ , Y_{new})$
}

Table 2.3: Foraging ant case study, solution c)

c) Q = {carrying_nothing, carrying_seed}
M = ((X_{curr} × Y_{curr}) × (X_{base} × Y_{base}) × Carrying_seed) where X_{curr}, Y_{curr}, X_{base}, Y_{base} = $\mathbb{Z}$, Carrying_seed = {$seed_1$, $seed_2$, ... $seed_n$} ∪ nil, n ∈ $\mathbb{N}$
m_o = ((2, 3), (0, 0), nil)
q_o = {carrying_nothing}
Σ = ((X_{new}, Y_{new}), Seed_Id), where X_{new}, Y_{new} = $\mathbb{Z}$, Seed_Id = {$seed_1$, $seed_2$, ... $seed_n$} ∪ {nil}, n ∈ $\mathbb{N}$
Γ = {"ant keeps moving empty", "ant detected and picked seed", "ant searches for base", "ant found base and left seed"}
Φ = {
search_and_see_seed (((X_{new}, Y_{new}), Seed_Id), ((X_{curr}, Y_{curr}), (X_{base}, Y_{base}), Carrying_seed)) = ("ant detected and picked seed", ((X_{new}, Y_{new}), (X_{base}, Y_{base}), Seed_Id)) if (X_{curr}, Y_{curr}) ≠ (X_{base}, Y_{base}) ∧ (X_{curr}, Y_{curr}) ≠ (X_{new}, Y_{new}) ∧ Carrying_seed = nil ∧ Seed_Id ≠ nil,

search_for_seed (((X_{new}, Y_{new}), Seed_Id), ((X_{curr}, Y_{curr}), (X_{base}, Y_{base}), Carrying_seed)) = ("ant keeps moving empty", ((X_{new}, Y_{new}), (X_{base}, Y_{base}), Carrying_seed)) if (X_{curr}, Y_{curr}) ≠ (X_{base}, Y_{base}) ∧ (X_{curr}, Y_{curr}) ≠ (X_{new}, Y_{new}) ∧ Carrying_seed = nil ∧ Seed_Id = nil,

search_for_base (((X_{new}, Y_{new}), Seed_Id), ((X_{curr}, Y_{curr}), (X_{base}, Y_{base}), Carrying_seed)) = ("ant searches for base", ((X_{new}, Y_{new}), (X_{base}, Y_{base}), Carrying_seed) if (X_{curr}, Y_{curr}) ≠ (X_{base}, Y_{base}) ∧ (X_{curr}, Y_{curr}) ≠ (X_{new}, Y_{new}) ∧ Carrying_seed ≠ nil,

leave_seed_at_base (((X_{new}, Y_{new}), Seed_Id), ((X_{curr}, Y_{curr}), (X_{base}, Y_{base}), Carrying_seed)) = ("ant found base and left seed", ((X_{new}, Y_{new}), (X_{base}, Y_{base}), nil) if (X_{curr}, Y_{curr}) = (X_{base}, Y_{base}) ∧ (X_{curr}, Y_{curr}) ≠ (X_{new}, Y_{new}) ∧ Carrying_seed ≠ nil
}

2.5 Membrane computing modelling approaches

The structure of a biological living cell, the system on which a living cell operates and the functions of their membranes, gave a motivation to a whole new area of research, namely *membrane computing* [75, 76]. The first membrane computing models built, were comprised of *membranes* arranged in a hierarchical structure (in a way that they can be included one inside another), resembling the membranes of living cells. These membranes draw up the boundaries of the compartments (i.e. regions), yet allowing for developing of chemicals within them [76]. Such operations are defined through *evolution rules* and the multisets of chemicals are actually named as *objects*. There are basically two types of evolution rules in these models: rewriting rules (used for modeling chemical reactions) and communication rules (rules that define a function for passing objects through the membranes) [75, 76].

There are many advances in membrane computing up to date. Starting with a hierarchical structure of the membranes that resemble a tree-like cell arrangement, now we have models with a tissue-like (or graph) membranes structure or even more advanced neural-like membrane systems, inspired from neurobiology and neural networks [76]. The computing mechanisms found in membrane computing in general, are called *P systems*. Fig. 2.6 shows a graphical representation of a P system which outputs square numbers.

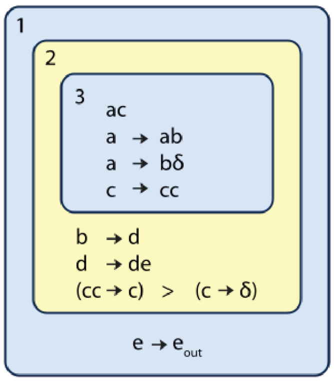

Figure 2.6: A representation of a P system which outputs square numbers.

Even though this research area is new, today there are large number of studies targeting P systems, in variety of application areas (biology, bio-medicine, economics, computer science, etc). The classical P systems (described earlier as the first membrane computing model) are found to experience certain obscurity when it comes to modelling some areas, such as MAS with a dynamic configuration [87]. Therefore, there are many different definitions of P systems, each of them aiming to specific domain but having different computation. Some of the other more prominent types of P systems are [77]:

- catalytic P systems;
- communication P systems;
- P systems with string objects;
- P automata;
- splicing P systems;
- tissue P systems;
- population P systems;
- kernel P systems;
- P systems with active membranes, and many others.

An interesting observation is that most of these different P systems definitions, are actually equivalent to Turing machines, therefore they are computationally complete [76]. The classical P systems for instance, apply their rules in parallel for each cell, allowing computation of NP-complete problems in linear time [4]. This work will not focus on defining each of the types of P systems but will draw focus on population P Systems.

2.5.1 Population P systems

A *population P System* (PPS [4]) is a collection of different types of cells that can evolve according to specific rules and exchange substance with the neighbouring cells. The rules can manage communication among cells, cell division, cell death, etc. Formally, a PPS is described in Definition 2.4 [4], and an abstract example of a PPS is given in Fig. 2.7.

Definition 2.4. A *PPS* can be defined as P = (V, K, γ, α, w_E, C_1, C_2, . . . , C_n, R), such that:

- V is a collection of all the objects from all the cells within the system, $V = \{object\ a,\ object\ b,\ ...\}$;
- K is a collection of all the different types, associated with each individual cell in the system, $K = \{t_1,\ t_2,\ ...\}$;
- γ is the initial structure of the undirected graph, formally defined as:
  $\gamma = (\{1,\ 2,...\ n\},\ A)$, with $A \subseteq \{\{i,\ j\} \mid 1 \leq i \neq j \leq n\}$;
- α is a finite set of bond making rules $(t,\ x_1;\ x_2,\ p)$, such as $x_1,\ x_2 \in V^*$ and $t,\ p \in K$;
- $w_E \in V^*$ is the multiset of objects initially assigned to the environment;
- $C_i = (w_i,\ t_i)$ is a tuple that contains a finite multiset of objects $w_i \in V^*$ and a type $t_i \in K$, for each $1 \leq i \leq n$;
- R is a finite set of cell evolution rules.

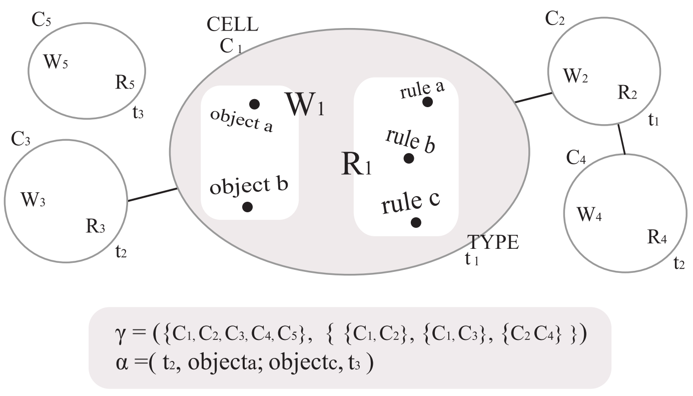
C5
W5
R5
t3
CELL
C1
W1
object a
object b
R1
rule a
rule b
rule c
TYPE
t1
C2
W2
R2
t1
C3
W3
R3
t2
C4
W4
R4
t2
γ = ({C1, C2, C3, C4, C5}, { {C1, C2}, {C1, C3}, {C2 C4} })
α =(t2, objecta; objectc, t3)

Figure 2.7: An abstract example of a Population P System.

Each node from the graph in Fig. 2.7 represents a membrane and is referred to as a *cell*.

Definition 2.5. Every *cell, C_i,* contains a multiset of *objects, W_i,* which represent some kind of a property (for instance, objects could be *age*, *temperature*, *position*, etc). Objects are represented in the form (*label:value*), where *label* is a descriptor of the object, and *value* holds the actual object's value (for instance (*age*:$\mathbb{N}_0$) or (*temperature*:$\mathbb{R}$)). A cell is associated with a type t_i and all cell types in the system are defined as K.

Every cell contains rules for evolution, R_i (a, b, $c \in V^*$ and t, $p \in K$):

- Communication rules – Communicating objects through the cells' boundaries. These rules could be in one of the forms: $(a;\ b,\ in)_t$, $(a;\ b,\ enter)_t$ and $(b,\ exit)_t$ as presented in Fig. 2.8, i.–iii. accordingly. These rules manipulate the objects within two cells, based on their types and the existing bonds, or they manipulate the objects between a cell and the environment (again, based on the type of the cell).
- Transformation rules – Modifying/rewriting of the objects within a cell. These rules have the form $(a \rightarrow b)_t$ and they allow an object to be transformed to a different one within a cell of a certain type. They are presented in Fig. 2.8, iv.
- Differentiation rules – Changing of the cells' types. They have the form $(a)_t \rightarrow (b)_p$ and are presented in Fig. 2.8, v.
- Division rules – Generating new cells. These rules can be presented as $(a)_t \rightarrow (b)_t\ (c)_t$ as seen in Fig. 2.8, vi.
- Death rules – Removing cells from the system. They have the form $(a)_t \rightarrow \dagger$ as presented in Fig. 2.8, vii.

It can be observed that the first two types of rules (communication and transformation rules) affect the system on a micro level, i.e. the specific properties of each individual cell. Thus at every cycle, all the applicable transformation and communication rules are applied, also known as *maximal parallelism* in the use of rules. On the other hand, the differentiation, division and death rules affect the system on a macro-level, i.e. they affect the system's structure. Therefore, at every cycle only one rule from all the applicable ones is non-deterministically selected and applied.

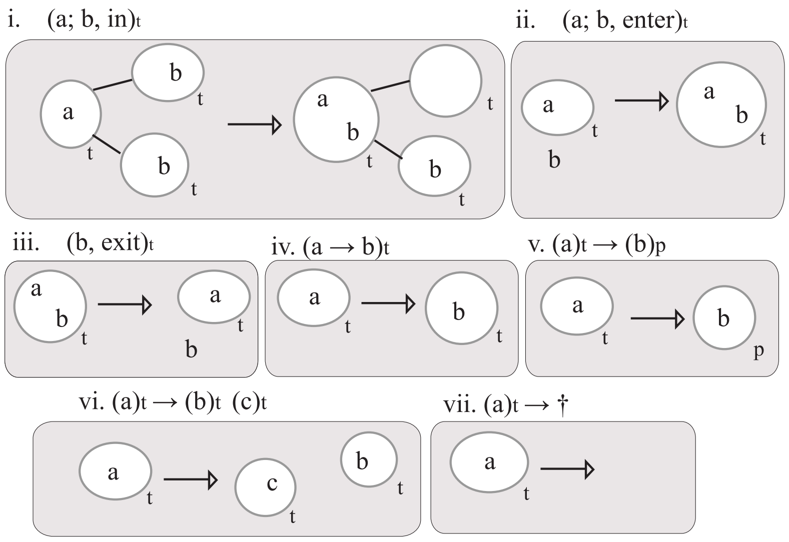

Figure 2.8: Evolution rules in population P system with active cells.

Besides the cell's rules for evolution, a PPS with active cells contains a different type of rules as well, which in turn affect the bonds between the cells. These are called bond-making rules and they define which cells are able to communicate with each other. These rules are applied last, at every step of execution (cycle). The representation $(t,\ x_1;\ x_2,\ p)$ can be interpreted as: a new bond between all the pairs of cells $(t,\ p)$ with types t and p can be formed, `iff` the object x_1 belongs to the cell of type t, and the object x_2 belongs to the cell of type p.

2.5.2 Spatial agent modelling with PPS

PPS with active cells exhibit certain shortcomings when it comes to spatial modelling. For the purpose of presenting them, let us consider a case study inspired from a simple ant colony:

Case Study 2.2. ***Ant lines.*** We anticipate to model the behaviour of the ants within a colony, given that there is one *leader ant* moving towards a source of food, and every other ant is following the leader. At the beginning, all of the ants are in their nest. With the first evolution, the leader leaves the nest first, and chooses a random path to follow. Subsequently, one of the other ants leaves the nest as well, headed towards the current position of the leader. After a while, the second ant leaves the nest, and follows the ant ahead of it in the same manner. This process repeats until all of the ants reach the food source.

It might be observed that this example looks like flocking. Utilizing PPS with active cells, this case study can be modelled as presented in Table 2.4. It should be noted that Table 2.4 contains a specification written with the PPS Description Language (PPSDL [86]) explained in Section 3.3.

This case study represents a phenomenon that could be modelled by employing the object-based parallel modelling power of PPS with active cells. The difficulties encountered in terms of modelling the spatial characteristics of the ants though, can be summarised as follows:

- There is a lack of supporting functions to describe movement. Functions, like movement to a random/specific position or heading to a random direction, are very commonly found in the biological systems in the nature and represent a common characteristic to every spatial system, as well.

- The position and the direction are not natively supported (we refer to predefined object types), and these properties are also common when it comes to spatial modelling.

- The concept of *sensing* cannot be described with the existing communication rules. In other words, abstract information (objects that cannot be quantified) are normally communicated by multiplication, i.e. perception which leaves multiple copies to both parties.

Furthermore, one might argue that having non-deterministically chosen cell evolution rules, happens in the biological systems and might represent one of the most important factors that leads to emergent behaviour. However, this is not true for all of the biological processes. Therefore, an interesting property that can be appended to this notion is prioritising of behaviours [43]. Referring to the case study, let us assume that there are some obstacles in the environment (for instance, water holes or fire places) and the ants are naturally (by instinct)

Table 2.4: Modelling the ant lines case study with PPS

$V = \{(\mathit{nest\_position}\text{: } \mathit{Position}), (\mathit{food\_position}\text{: } \mathit{Position}), (\mathit{leader\_heading}\text{: } \mathit{Direction}), (\mathit{ant\_position}\text{: } \mathit{Position}), (\mathit{ant\_heading}\text{: } \mathit{Direction}), (\mathit{timer}\text{: } \mathbb{N}_0), (\mathit{ant\_no}\text{: } \mathbb{N})\ \}$, where $\mathit{Position} \in \mathbb{N}_0 \times \mathbb{N}_0$ and $1 \leq \mathit{Direction} \leq \mathit{360}$;

$K = \{\mathit{leader\_ant}, \mathit{follower\_ant}\}$;

$\gamma = (\{1, 2, 3, 4, 5, 6\}, \{\ \})$;

$\alpha = (\mathit{leader\_ant}, (\mathit{timer}\text{: } 1); (\mathit{timer}\text{: } 10), \mathit{follower\_ant}, \mathit{follower\_ant}, (\mathit{timer}\text{: } 1); (\mathit{timer}\text{: } \mathit{ant\_no} * 5), \mathit{follower\_ant})$;

$w_E = \{(\mathit{nest\_position}\text{: } (0, 0)), (\mathit{food\_position}\text{: } (0, 50))\}$;

$C_1 = (\{(\mathit{ant\_position}\text{: } (0, 0)), (\mathit{timer}\text{: } 5), (\mathit{ant\_no}\text{: } 1), (\mathit{leader\_heading}\text{: } 60)\}, \mathit{leader\_ant})$, and

$C_i = (\{(\mathit{ant\_position}\text{: } (0, 0)), (\mathit{ant\_heading}\text{: } 60), (\mathit{timer}\text{: } 5 * i\ ), (\mathit{ant\_no}\text{: } i)\ \}, \mathit{follower\_ant})$, for $2 \leq i \leq \mathit{6}$;

$R_1 = \{\ (\mathit{timer} \rightarrow \mathit{timer}\ \text{-}\ 1)_{\mathit{leader\_ant}},$
$(\mathit{ant\_position})_{\mathit{leader\_ant}} \rightarrow \dagger$ if $\mathit{ant\_position} = \mathit{food\_position}$,
$(\mathit{leader\_heading} \rightarrow \mathit{RandHeading})_{\mathit{leader\_ant}},$
$(\mathit{ant\_position} \rightarrow \mathit{RandPosition})_{\mathit{leader\_ant}}\},$
where *RandHeading* and *RandPosition* are functions that return a random direction and position accordingly, and

$R_i = \{\ (\mathit{timer} \rightarrow \mathit{timer}\ \text{-}\ 1)_{\mathit{follower\_ant}},$
$(\mathit{ant\_position})_{\mathit{follower\_ant}} \rightarrow \dagger$ if $\mathit{ant\_position} = \mathit{food\_position}$,
$(\mathit{ant\_position} \rightarrow \mathit{RandPosition})_{\mathit{follower\_ant}}$ if $\mathit{timer} = 0$,
$(\mathit{ant\_heading} \rightarrow \mathit{Heading})_{\mathit{follower\_ant}}\ \},$
for $2 \leq i \leq \mathit{6}$ where *RandPosition* and *Heading* are functions that return a random position and the heading of the previous ant, accordingly.

constructed to avoid these obstacles. This would point to the fact that not all of the processes in nature are of equal importance, and there is a need to model some specific subset of rules in a way that they will be deterministically executed. Driven by the subsumption architecture [7], the work presented in [43] suggests defining a partial ordering over the evolution rules found in PPS with active cells.

2.6 Hybrid modelling approaches

The OPERAS framework for formal modelling originated as a combination of formal methods, or even transforming to one another, for the purpose of achieving their complementary features [88]. Formally, it is defined in Definition 2.6 [89, 88].

Definition 2.6. OPERAS is a tuple (O, P, E, R, A, S) containing:

- O is a set of reconfiguration operations or rules, which defines how the system structure evolves;
- P is a set of percepts for the agents, which in essence is the set of valid inputs to the system;
- E is the initial configuration of the environments model, ir describes the elements that are present and the agents can perceive or affect;
- R is a relation that defines the existing communication channels, which are the existing channels among agents;
- A is a set of participating agents; and
- S is a set of definitions of types of agents.

The rules in the set of reconfiguration operations O, have a general form *condition* ⇒ *action*. Operators are applied with each operation and they create or remove a communication channels between agents. These operators can also introduce agents in the system, or remove existing agents from the system [88].

This framework views an agent as composed of two parts [88]:

- Individual agent behaviour, and

- Dynamic structural mutation.

The aim of OPERAS however, is not only decomposing of the modelling process. It is flexibility to allow describing the agent's parts with different formal methods in the choice of the modeller. This opens new possibilities towards biologically-inspired models.

There can be different instantiations of OPERAS, one of which is the OPERAS$_{XC}$ which employs communicating X-machines and ideas from PPS [89, 88]. This instantiation of OPERAS aims at combining their advantages and suppressing their weaknesses.

In OPERAS$_{XC}$ the behaviour of the system (individual agents) are modelled as a CXM, and the structural mutation mechanism of each agent is a PPS (see Fig. 2.9) [88]. The agents' communication is dealt with the CXM level.

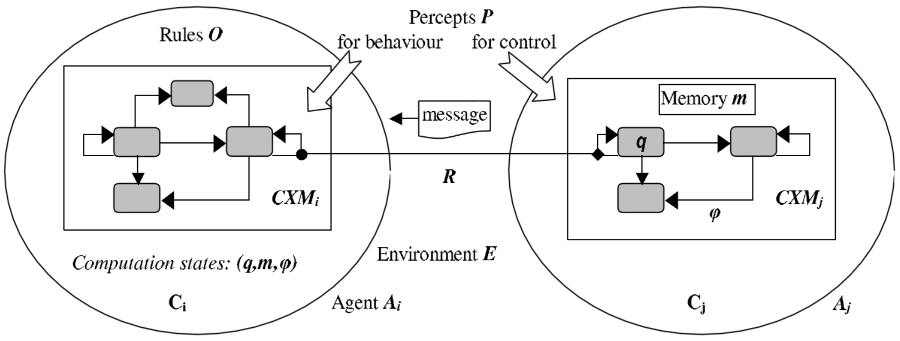

Figure 2.9: An abstract example of a OPERAS$_{XC}$ consisting of two agents.

2.7 Comparison of modelling approaches

All the formalisms described in the previous sections are characterised with certain advantages and disadvantages regarding modelling and implementation of bio-MAS, as summarised in Table 2.5. XM and CXM are particularly suitable at modelling the internal states of an agent and the agent's perception [31]. On the other hand, they do not support dynamic system reconfiguration (birth and death of agents). On the other hand, PPS support rules for reconfiguration of the structure within the system, but it is hard to distinguish different attributes of an agent [75].

Table 2.5: Advantages and disadvantages in modelling with XM, CXM, PPS and OPERAS

Advantages	**Disadvantages**
XM	
– Ability to model the internal states of an agent, the agent's perception, the agent's knowledge of the environment and how the agent can change its internal state and knowledge when a function is triggered.	– Do not support birth of agents (dynamically creating agents) and death of agents (dynamically destroying agents). – It is not easy to specify agents that move in space.
CXM	
– Offer a way to represent the static links and exchange of messages between agents.	– Reconfiguration of the structure within the system remains not achievable. – It is not easy to model the topology of MAS.
PPS	
– Support rules for reconfiguration of the structure within the system. – Deal with a dynamic system's structure (describing the behaviour of a system). – Support rules for cell birth and cell death (ability to reconfigure a system throughout its lifetime).	– It is hard to group objects based on their use (distinguish different attributes of an agent). – The objects are the only way with which an agent's internal state can be represented.
OPERAS	
– Contemplates into separating the behaviour of an agent from its control (modelling each of them individually). – Provides ground to formal description of the changes occurring in the structure of a dynamic MAS.	– Protocols for agent communication and interaction are not supported.

It can be concluded that the drawback of XM (ability to reconfigure a system throughout its lifetime) is supported in PPS. On the other hand, PPS lack the advantages found in XM. Finally, OPERAS$_{XC}$ serves the purpose of achieving their complementary features.

It is important to highlight that this comparison of formalisms is not exhaustive but focused on XM, CXM, PPS and OPERAS. The main reason for this choice is to target modelling formalisms that are easy to be learnt by modellers of different backgrounds having limited mathematical knowledge (such as biologists).

2.8 Summary

Parts of the following objectives were met with this chapter:

O1: Investigate on spatial systems, modelling formalisms for spatial bio-MAS outlining properties and disadvantages of existing modelling formalisms, as well as verification and simulation strategies and how can they be enhanced to better support complex spatial bio-MAS.

O2: Identifying illustrative case studies which are scalable (experiments with different numbers of agents), simple to be modelled and have spatial characteristics.

This research was published in [68] and [66].

Chapter 3

Description of languages for modelling and corresponding tools

There are many different tools for practical formal modelling of bio-inspired systems as spatial MAS. Referring back to the previous topic on different modelling approaches, the models of a X-machine can be described with tools such as: X-System or JSXM, and the PPS models with PPSDL. These tools will be introduced in this chapter because they are used for modelling different case studies later on. Moreover, an agent based modelling framework, namely FLAME, will also be introduced as a tool for modelling that employs XM as the basic computational model.

3.1 X-System

X-System is a tool created to support modelling with X-machines [42]. With this tool, the X-machine models can be specified in the X-machine Definition Language (XMDL [42]) and textual simulation. XMDL is a listing of definitions that matches the tuples of X-machine's definition. For example, a function in XMDL takes an input and a memory value, returning an output with a new memory value. Briefly, the XMDL keywords can be presented as [42]:

- `#model <model name>` – Assigns a name to a model.
- `#input <input tuple>` – Describes the input set and corresponds to Σ in XM.
- `#output <output tuple>` – Describes the output set and corresponds to Γ in XM.
- `#states <set of states>` – Defines the set of states and corresponds to `Q` in XM.
- `#memory <memory tuple>` – Defines the memory tuple and corresponds to `M` in XM.
- `#init state <state>` – Sets the initial state and corresponds to $\mathtt{q}_0$ in XM.
- `#init memory <memory value>` – Sets the initial memory and corresponds to $\mathtt{m}_0$ in XM.
- `#transition(<state>, <fun>) = <state>` – Defines each transition in F and corresponds to `F` in XM.
- `#fun <function definition>` – Defines a function in and corresponds to Φ in XM.

The description language of a Communicating X-machines model is named as XMDL_*c* [44, 46]. It represents an extension of the standard XMDL definition to allow definition of the models in the system and how their functions communicate.

Advantages of X-System are [42]:

- The XMDL model represents a list of definitions which is close to the mathematical notation used for the definition of XM;
- Comes with a parser that helps in identifying omission and logical errors of the specification;
- Comes with an animator and a model checker; and

- Demonstrated to be successful in facilitating formal development of complex real world problems [21].

On the other hand, the disadvantages are: it uses only ASCII characters, and the graphical interface of the integrated system needs to be finalised [42].

3.2 JSXM

The models of a X-machine can be also described with JSXM [18]. JSXM introduces a syntax for X-machines specifications based on XML and Java.

Some examples of the JSXM modelling language are provided as follows [18, 19]:

- The set of the initial state (corresponds to $\mathtt{q}_0$ in XM):

```
<initialState state="initial" />
```

- The set of states (corresponds to `Q` in XM):

```
<states>
        <state name="A" />
        <state name="B" />
</states>
```

- The memory and the initial memory (corresponds to `M` and $\mathtt{m}_0$ in XM):

```
<memory>
        <declaration> int C </declaration>
        <initial> C = 0 </initial>
</memory>
```

An example of a function can be presented as [18]:

```
<function name="D" input="d" output="dOut">
        <precondition>
                C > d.getE()
        </precondition>
```

```
                <effect>
                        C = C - d.getE();
                </effect>
        </function>
```

One of main advantages of JSXM is that it represents a tool for automated test generation [19]. Other advantages include [19]:

- In the case of input-uniform specifications, JSXM test generation is guaranteed to work;
- JSXM allows the definition of a CXM (each class is separately modelled as an XM); and
- JSXM are demonstrated to be applicable in several application scenarios: Web service testing, monitoring and run-time verification.

On the other hand, disadvantages are [19]:

- The input generators need to be changed such that test generation works for generic specifications; and
- Lacks of test generation based on state-counting (in order to cover cases of specifications in which not all states are separable).

These disadvantages however, have been identified as a future work by the authors [19].

3.3 PPS-System

The PPS models can be described with the PPS Description Language (PPSDL [86]) and textual simulation. PPSDL is characterized along the same lines as XMDL, but its list of definitions match the constructs of a PPS with active cells. PPSDL is an orthogonal, mark-up and strongly typed language. Briefly, the PPSDL keywords can be presented as [87]:

- `#model <model name>` – Assigns a name to a model and corresponds to `P` in PPS.

- `#object <obj name> = <obj type>` – Defines a type of object and corresponds to `V` in PPS.
- `#cell types = <set of types>` – Describes the cell types set and corresponds to `K` in PPS.
- `#graph = <set of cell pairs>` – Defines the graph and corresponds to γ in PPS.
- `#bond making rule <r name> <rule definition>` – Defines a rule and corresponds to α in PPS.
- `#env objects = <set of objects>` – Initial env. objects and corresponds to $\mathtt{w}_E$ in PPS.
- `#cell <cell name>:<set of objects> <cell type>` – Defines a cell and corresponds to $\mathtt{C}_i$ in PPS.

Moreover, the PPSDL keywords for the different types of rules (`R` in PPS) can be presented as [87]:

- `#transformation rule <r name><rule def>`
- `#communication in rule <r name><rule def>`
- `#communication exit rule <r name><rule def>`
- `#differentiation rule <r name><rule def>`
- `#division rule <r name><rule def>`
- `#death rule <r name><rule def>`

A PPSDL model is initially compiled to Prolog. The computation is then animated according to PPS theory [87]. Advantages of PPS-System are [87]:

- It is aimed to be an interchange language between tools developed around P-Systems; and
- It does not deviate from the PPS formal mathematical notation.

On the other hand, the disadvantages of PPS-System include [87]:

- It uses only ASCII characters;
- There is a need of a platform for thorough evaluation of the PPSDL notation; and
- There is a lack of a graphical display to visually show the model computation.

3.4 FLAME

FLAME (Flexible Large-scale Agent Modelling Environment) is an agent based modelling framework that employs XM as the basic computational model (every agent is a XM). Figure 3.1 illustrates a diagrammatic representation of the X-agent used by the framework, where S_i are the agent's states, and F_i are the transition functions that use the memory and input messages facilitating the agent to change a state.

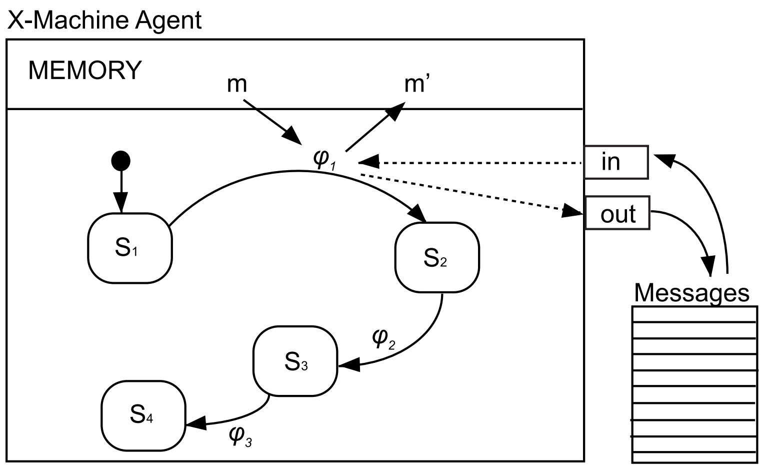

Figure 3.1: X-agent diagram.

FLAME is specialised to support mainly projects related to cell biology, including tissue cultures and signalling pathways, given that it initially originated from a project targeted on predicting the emergent behaviour of cells in epithelial tissues [83, 12]. The FLAME models can be described with XMML, XM markup modelling language (similar to XML tags) for defining agents and their communication links. On the other hand, the functions are implemented in the C programming language. Figure 3.2 shows the block diagram of the framework in which a model XMML file and a functions file are provided as inputs into a parser program (XParser) [93]. The inputs are converted into simulation code by the XParser which given starting values can simulate the model to produce results and animation.

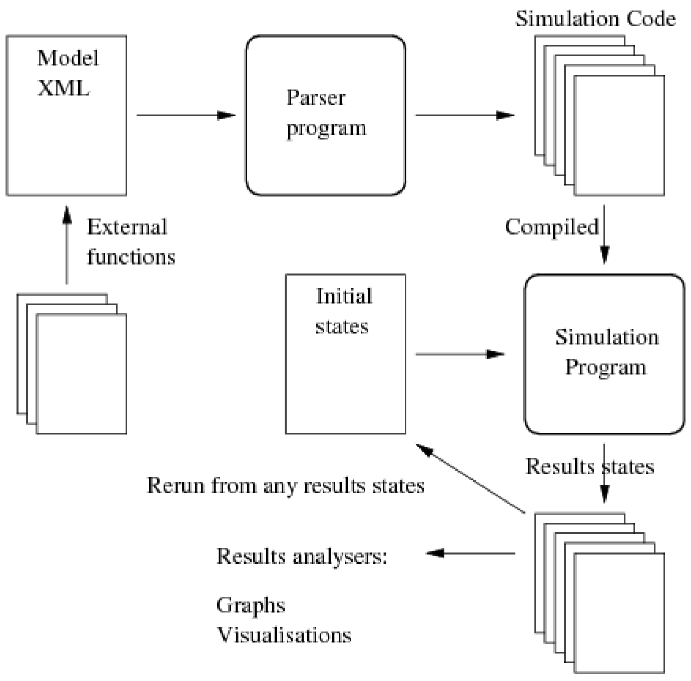

Figure 3.2: Block diagram of FLAME framework.

FLAME has the following advantages:

- Successful in modelling many biological systems as a consequence;
- Provides automatically parallelisable models;
- Allows high concentrations of agents to be simulated and the results to be achievable in finite time; and
- Functions are written in C and thus they are directly executable.

A disadvantage of FLAME could be the fact that it supports only fundamental data-types. Moreover, the functions cannot be written in any other language other than C, which requires a certain level of experience and expertise.

3.5 Summary

The advantages and disadvantages of the description of languages for modelling are summarised in Table 3.1.

This chapter contains parts of the following objective:

O1: Investigate on spatial systems, modelling formalisms for spatial bio-MAS outlining properties and disadvantages of existing modelling formalisms, as well as verification and simulation strategies and how can they be enhanced to better support complex spatial bio-MAS.

Table 3.1: Advantages and disadvantages of the description of languages for modelling and corresponding tools

Advantages	**Disadvantages**
X-System	
– The XMDL model represents a list of definitions which is close to the mathematical notation used for the definition of XM. – Comes with a parser that helps in identifying omission and logical errors of the specification. – Comes with an animator and a model checker. – Demonstrated to be successful in facilitating formal development of complex real world problems.	– Uses only ASCII characters. – The graphical interface of the integrated system needs to be finalised.
JSXM	
– It is a tool for automated test generation. – In the case of input-uniform specifications, the JSXM test generation is guaranteed to work. – Allows the definition of a CXM. – Demonstrated to be applicable in several application scenarios.	– The input generators need to be changed such that test generation works for generic specifications. – Lacks of test generation based on state-counting.
PPS-System	
– Semantics directly inherited by the PPS formal definition. – Aimed to become an interchange language between P-Systems tools.	– Lack of a graphical display for visual display of the computation. – Needs a platform to thoroughly evaluate the PPSDL notation. – Uses only ASCII characters.
FLAME	
– Successful in modelling many biological systems as a consequence. – Provides automatically parallelisable models. – Allows high concentrations of agents to be simulated and the results to be achievable in finite time. – Functions are written in C and thus they are directly executable.	– It supports only fundamental datatypes. – The functions cannot be written in any other language other than C, which requires a certain level of experience and expertise.

Chapter 4

Simulation and validation of bio-systems

Bio-systems exhibit an increased amount of complexity and this leads to difficulties in the process of modelling, as well as in ensuring the correctness on the model and its implementation. Looking at the framework for developing bio-MAS presented in Fig. 2.2, this chapter talks about step `IV`: formal verification (model checking), informal verification (visualisation or visual simulation) and testing. The focus falls to verification strategies of XM and P system models.

4.1 Introduction to verification, simulation and validation

Formal verification and *validation* are processes which employ formal methods to confirm that a model satisfies its requirements and specifications [5].

> **Definition 4.1.** *Verification* techniques, such as *model checking*, are applied to check whether a model satisfies certain properties, whereas *validation* is applied to confirm that a model satisfies the user requirements.

When it comes to spatial MAS, it is not always feasible to apply formal verification and validation techniques. This constraint rises from the fact that some of the properties found in these systems can have infinitely many values, which naturally leads to state space explosion.

> **Definition 4.2.** *Model checking* is one of the most widely used formal verification technique, which focuses on thorough exploration on a predetermined state space, trying to conclude whether some properties of a complex system are being met.

Basically, a model checker accepts as an input the model (as a labelled transition diagram) and a property defined with temporal logic. Then the checker would either verify that the given property is true, or will provide a counterexample, by following a specific search strategy on the labelled transition diagram.

In general, formal verification (such as model checking) of spatial MAS is an extremely complex task. On one hand is the fact that it leads to combinatorial explosion, but also the fact that all of the system's properties should be identified first before there is an attempt to be verified [68]. This notion is not always straightforward. As an example, properties and behaviour related to the positioning in space (or emergent behaviour such as line formation, flocks, schools, herds etc.) may occur in a system even though they were not explicitly modelled. This leads to the fact that spatial MAS may exhibit behaviour that was not present in original system's requirements. In order to detect such characteristics in systems, *simulation tools* providing visual output should be applied. To summarize:

- Visual simulation can serve as an informal verification method for systems that have spatial characteristics, which in turn are not formally verifiable; and

- Visual simulation can help into discovering emergent properties, which are common in spatial MAS.

Having state-based agent models as a starting point, formal methods can be used to model spatial agents. Our aim is to find the most suitable way to visualise the behaviour of such models by using a simulation tool that will meet certain criteria and facilitate the process of refining the formal model to executable code. There exist large-scale MAS simulation platforms that support

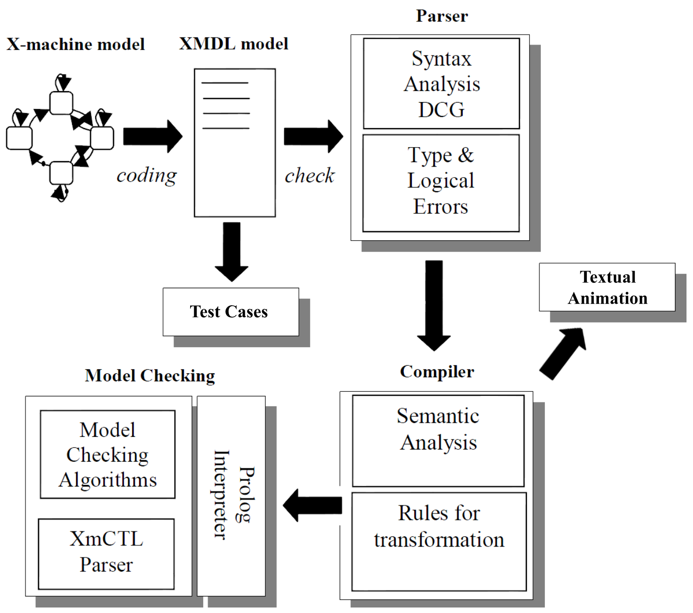

Figure 4.1: Verification and validation of an X-machine.

formal state-based models [81], such as: Repast [61], Swarm [58], NetLogo [95], Mason [54], and Flame [12, 72], to name a few. Such platforms may facilitate the informal verification of a spatial model by allowing researchers to compare the simulation's outcome with the expected behaviour of the system or to discover emergence.

4.2 Verification of XM models

XMs are supported by formal verification strategies [34, 20] as shown in Fig. 4.1. XMDL is facilitated with a parser built using Definite Clause Grammars (DCG) notation [45]. Apart from the syntax errors, this outputs warnings of any kind of syntax error or omission [45]:

- "State defined is not used in transitions";
- "The X-Machine is non-deterministic";
- "User types are not defined"; etc.

Examples of errors are:

- “The initial memory tupple arity is different from memory type”;
- “Function in transition is not defined”;
- “Cannot infer the type of variable”;
- “Memory parameter inconsistent with memory”; etc.

The *semantic analysis* and the *rules for transformation*, are being checked by the *compiling component*. This is facilitated by defined rules under which the specification is translated into the equivalent Prolog code. This Prolog code is after utilised by an animation tool, which simulates the computation of an X-machine.

The *model checking* component in Fig. 4.1 defines a new logic, XmCTL [20]. With the implementation of model checking algorithms, this component can determine whether a property is true or false. Finally, XMs are also supported with automatic generation of *test cases*, which is proved to find all faults in the implementation [34].

Recent research [19] employs JSXM as tool for automated test generation for XM. Basically, JSXM facilitates [19]:

- Animation of XM models for the purpose of model validation;
- Automatic generation of abstract test cases from the XM specifications; and
- Transformation of abstract test cases into concrete test cases (in the implementation language of the system).

4.3 Verification of P system models

There are several approaches towards verification and testing strategies for P systems. A recent model checking-based approach [37] for verification a P system model, utilises the NuSMV symbolic model checker [13]. NuSMV verifies the correctness of the system using temporal logic formulae and if the specified properties are false, counterexamples are provided. This feature is used for the creation of test cases. Similar work in [39], suggests a P system model verification using the Spin model checker [33]. It provides techniques to translate a P system into Promela which is the modelling language for Spin.

Other work in [36], utilises the Event-B (a formal modelling language [1]) for modelling, verification and testing of P Systems. Verifying and testing of P systems are based on ProB, the model-checker of Event-B. Different approach for testing P systems described in [23], defines a grammar and finite state machine based strategies (approach focussed on cell-like P systems, but applicable for tissue-like P systems as well). Finally, P systems could also be translated to a XM model in order to apply the XM verification and testing strategies [48].

To elaborate on the verification and testing strategies, the first approach that utilises the NuSMV symbolic model checker can be explained in more details, namely [37]:

1. The P system is transformed into a Kripke structure [49];
2. The Kripke structure is written in NuSMV syntax;
3. These formulae are negated (for instance, *this rule will never be applied*);
4. NuSMV is run and the counterexamples provided for the negated formulae are interpreted as test cases.

4.4 Visual simulation: case study and discussion

The following problem can be highlighted:

> *Utilizing XM or P systems for modeling spatial systems, needs an exponential time to complete the execution of a model checker. This is due to the thorough exploration on the system's state, which means all possible positions (coordinates) and directions.*

In order to explain these concepts, let us consider the following case study [6]:

Case Study 4.1. ***Aggressor-Defender.*** The following case study, known as the aggressor-defender game [6], consists of two groups of agents, namely defenders (or *friends*) and aggressors (or *enemies*). Agents follow one of two strategies:

- defending: in each cycle the agent tends to position itself between a friend and an enemy (such as they defend the friend against the enemy; see Fig. 4.2 a).
- fleeing: in each cycle the agent tends to position itself so that a friend is between it and an enemy (such as the friend protects them from the enemy; see Fig. 4.2 b).

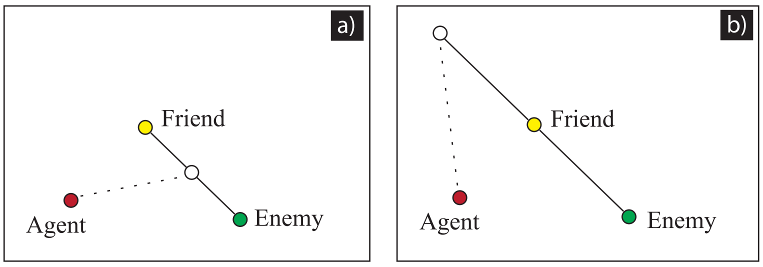

Figure 4.2: Rules for playing the aggressor-defender game. a) The defend behaviour. b) The flee behaviour.

This particular case study was chosen for reasons of exposition because of the following characteristics:

- it has spatial characteristics and therefore it can be modelled as a spatial MAS;
- it is scalable as users can experiment with different numbers of agents;
- it exhibits an emergent behaviour not evident in its definition;
- it is very simple to be modelled, which demonstrates that visualisation is helpful even for the simplest spatial MAS; and
- it is general enough to make safe conclusions that relate to the characteristics and the suitability of visual simulations platforms.

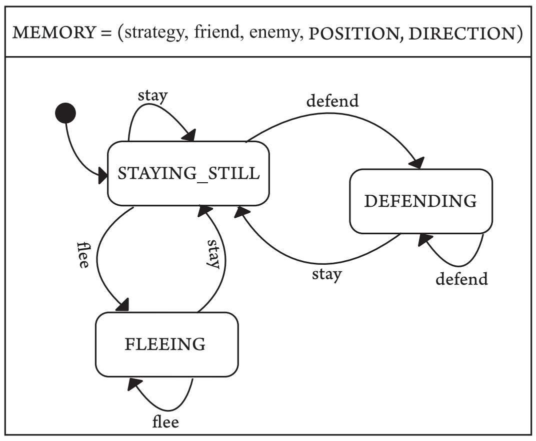

Figure 4.3: X-Machine model of the aggressor-defender game.

The XM state transition diagram of the *Case Study 4.1: Aggressor-Defender* is provided in Fig. 4.3.

As an instance, the function *defend* can be formally modelled with the XM approach as follows:

$$defend(((x_{friend}, y_{friend}), (x_{enemy}, y_{enemy})),$$
$$(strategy, friend_{id}, enemy_{id}, (x, y), direction)) \mapsto$$
$$(((x\prime, y\prime), direction), (strategy_{new}, friend_{id}, enemy_{id},$$
$$(x\prime, y\prime), direction)), wherestrategy_{new} \leftarrow defending \wedge$$
$$x\prime \leftarrow (x_{friend} + x_{enemy})/2 \wedge y\prime \leftarrow (y_{friend} + y_{enemy})/2$$

At this point, the model can be translated into an executable form for a simulation platform. *Complete model* of the case study in a visual simulation platform, namely NetLogo, is presented in Appendix C. The visual output of the aggressor-defender game is provided as follows:

- Fig. 4.4 a) shows the simulation output when all the agents defend;
- Fig. 4.4 b) shows the simulation output when all the agents flee; and
- Fig. 4.5 shows three different simulation outputs when some agents defend and some agents flee.

In the simulation in Fig. 4.4 and 4.5, there is an observable emergent spatial behaviour, such as:

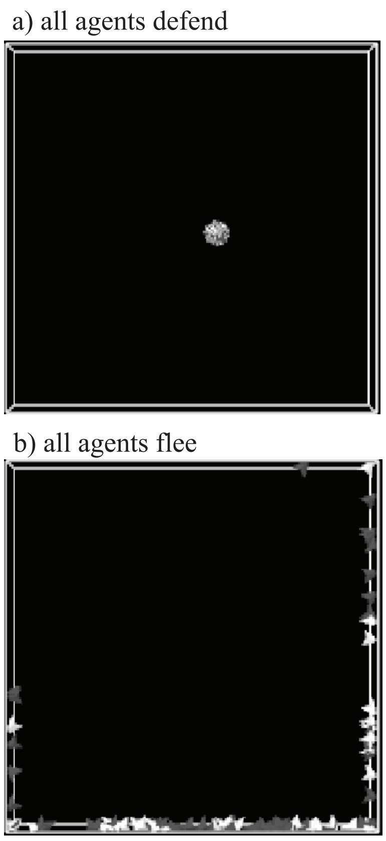

Figure 4.4: Visual output of the aggressor-defender game. a) All agents defend. b) All agents flee.

- The model in which all the agents defend behaved as the agents quickly collapsed into a tight knot, see Fig. 4.4 a).
- The model in which all the agents flee behaved as a highly dynamic group that expands over time towards the ends of the environment, see Fig. 4.4 b).
- The model in which some agents defend and some agents flee, exhibited three different emergent behaviours:
  - All agents collapsed into a knot (similar to the model in which all the agents defend) with the difference that this knot was now oscillating around the environment, see Fig. 4.5, i.

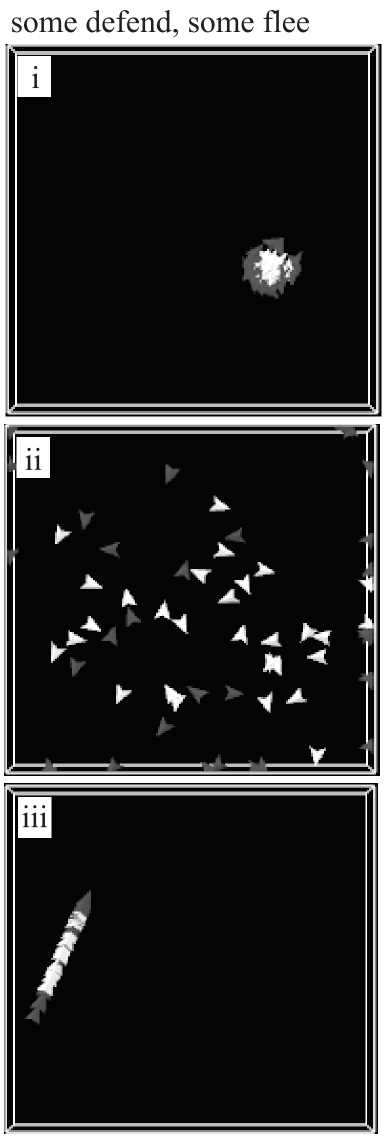

Figure 4.5: Visual output of the aggressor-defender game: three different outputs when some agents defend and some flee.

- The agents were stationary, randomly distributed and oscillating, see Fig. 4.5, ii.
- The agents would form a flocking, see Fig. 4.5, iii.

All of these different behaviours of the system, became apparent after *observing repetitive patterns from the visual simulation*. If the model was to be formally verified before simulation, *these emergent properties would probably not be discovered*. Moreover, *it is impossible to apply model checking techniques when it is not known what property to check for*. This example clearly demonstrates that *suitable simulation tools for modelling spatial MAS can help to detect unknown spatial behaviour, i.e. emergence*.

There were number of experiments performed on the aggressor-defender case study. Table 4.1 shows some of the results. The time is measured in ticks, which are a unit of time measurement inside the NetLogo simulator. The time was recorded at the point in which all of the agents were expanded towards the end of the environment (in the flee strategy), or all of them were collapsed into knot (in the defend strategy).

Table 4.1: Experiments on the aggressor-defender case study

Strategy	**Number of agents**	**Emergence**	**Time (in ticks)**
Flee	5	Expands over time	226
Flee	50	Expands over time	1490
Flee	100	Expands over time	1601
Flee	200	Expands over time	1452
Flee	500	Expands over time	1469
Flee	1000	Expands over time	1597
Flee	2000	Expands over time	2417
Defend	5	Collapsed into knot	166
Defend	50	Collapsed into knot	237
Defend	100	Collapsed into knot	247
Defend	200	Collapsed into knot	231
Defend	500	Collapsed into knot	260
Defend	1000	Collapsed into knot	280
Defend	2000	Collapsed into knot	257

4.5 Summary

This chapter covers some parts of following objectives:

O1: Investigate on spatial systems, modelling formalisms for spatial bio-MAS outlining properties and disadvantages of existing modelling formalisms, as well as verification and simulation strategies and how can they be enhanced to better support complex spatial bio-MAS.

O2: Identifying illustrative case studies which are scalable (experiments with different numbers of agents), simple to be modelled and have spatial characteristics.

Parts of this research were published in [68] and [66].

Chapter 5

Tools for Simulation

This chapter presents a survey of visual simulation platforms which are suitable for spatial state-based MAS models: NetLogo, Repast and the Flame visualiser. The comparison of these simulation platforms is based on the several criteria, namely: modelling, implementation, validation, verification, testing, visualisation, supported size of MAS and high performance computing.

5.1 Simulation platforms

NetLogo [95] is a simulation platform for multi-agent systems. It is supported by a functional language that can represent an agent's behaviour, as well as by an environment for the creation of a graphical user interface. As a programmable modelling environment, NetLogo is specialised in simulating natural and social phenomena, including modelling of complex systems [95, 94]. NetLogo supports an agent's actions called *commands*, and functions that compute and report results, *reporters*. The environment comes with many built-in commands and reporters called *primitives*, but the modeller is also allowed to define their own, called *procedures*. Furthermore, it supports custom defined agent and/or global variables, along with the built-in agent variables like the agent coordinates and heading.

Repast (REcursive Porous Agent Simulation Toolkit) [61, 14], a framework for creating agent-based simulations. It is composed of library of classes (i.e.

programs) which enhance the creation, running and display of MAS models. This system comes with a graphical user interface which can provide outputs in the form of a histogram, visual display of the interactions between the agents, or a chart of time-series data. This platform allows the specification of the logical and spatial structure of a model, the types of agents a model is composed of, and the individual properties and behaviour of the agents themselves. In this work we are interested in its latest release Repast Simphony. Repast provides visual point-and-click tools for model execution, as well as visualization and storage of the results. Automated results analysis, data mining and statistical analysis tools are also included [61].

Interesting to note is that Flame models can be visualized natively with the use of an external tool which comes with Flame, i.e. the Flame visualiser. Moreover, there is an extension to the Flame framework, Flame GPU, which is a high performance Graphics Processing Unit (GPU) [80]. Flame GPU models can be visualised in real time.

5.2 Comparison of simulation platforms

This section provides a comparison between NetLogo, Repast and Flame based on the following criteria:

- Modelling: diagrammatic or declarative state-based modelling. Diagrammatic models can be built by utilizing, for instance, flowcharts. Declarative models can be built with the use of a description language.
- Implementation: coding with the use of a programming language (functional, object oriented, scripting, etc.).
- Validation: a process used to demonstrate that *the model is built right* and that it satisfies its intended use when placed in its intended environment.
- Verification: a process used to confirm that *the model is correct* with respect to its requirements. One approach of formal verification is model checking, a thorough exploration on a predetermined state space, trying to conclude whether some properties of a complex system are being met.

- Testing: a dynamic technique for providing a series of inputs and comparing the respective outputs with the documented specification.
- Visualisation: visual output of the simulation platform.
- Supported size of MAS and *High Performance Computing* (HPC).

These factors are considered to be the most important when it comes to the visualization of spatial MAS. However, other factors such as related tools for development, interface, documentation, ease of use, support, open source code, extensibility, and GUI creation, should not be excluded. For the purposes of the comparison, *Case Study 4.1: Aggressor-Defender.* was implemented in all NetLogo, Repast and Flame environments. Table 5.1 demonstrates the function *defend* and Tables 5.2 and 5.3 show the description of the model.

Table 5.1: Aggressor-defender game: The function *defend* in NetLogo, Repast and Flame

FUNCTION defend in **NetLogo**

```
to defend
 facexy ([xcor] of friend +
  [xcor] of enemy) / 2
    ([ycor] of friend +
  [ycor] of enemy) / 2
end
```

FUNCTION defend in **Repast Simphony (ReLogo)**

```
def defend() {
 facexy(({ xcor }.of(friend) +
  { xcor }.of(enemy)) / 2,
    ({ ycor }.of(friend) +
  { ycor }.of(enemy)) / 2)
}
```

FUNCTION defend in **Flame**

```
  double handle_defend_X(double friendX, double enemyX)
  { double newPosition = (friendX + enemyX)/2;
    return newPosition;}

  double handle_defend_Y(double friendY, double enemyY)
  { double newPosition = (friendY + enemyY)/2;
    return newPosition;}
```

Table 5.2: Aggressor-defender game: Definition of the MAS in NetLogo and Repast

Description of the model in **NetLogo**

```
to setup
 clear-all
 create-turtles no-of-turtles
 ask turtles [
  set xcor random-xcor
  set ycor random-ycor
  set color one-of [red blue]
  set friend one-of other turtles
  set enemy one-of other turtles
 ]
end
```

Description of the model in **Repast Simphony (ReLogo)**

```
def setup() {
 clearAll()
 createTurtles(noOfTurtles)
 ask (turtles()){
  xcor = randomXcor()
  ycor = randomYcor()
  color = oneOf([red(),
    blue()])
  friend =
    oneOf(other(turtles()))
  enemy =
    oneOf(other(turtles()))
 }
}
```

5.2.1 Modelling

When it comes to defining agents and their communication links, NetLogo and Repast are weak in terms of modelling methodology. Instead, they provide an implementation language with which the user is expected to code the desired system. The declarative part of Flame enforces the modeller to think in terms of design. This represents a clear advantage of this framework.

X-machine models can be easily converted to Flame for the purpose of visual simulation, because every agent in Flame is an X-machine. The Flame framework is composed of X-agents, which have: states, memory, transition functions that use a memory, and input messages facilitating the agent to change a state. An example of the memory declaration can be found in Table 5.2 and Table 5.3, and the functions definition in Table 5.4. Finally, Repast Simphony models can be also built by utilizing point-and-click flowcharts.

Table 5.3: Aggressor-defender game: Definition of the MAS in Flame

Description of the model in **Flame**

```
<agents>
 <xagent>
  <name>FriendEnemy</name>
  <memory>
   <variable><type>int</type>
    <name>agent_id</name>
    <description></description></variable>
   <variable><type>double</type>
    <name>agent_x</name>
    <description></description></variable>
   <variable><type>double</type>
    <name>agent_y</name>
    <description></description></variable>
   <variable><type>int</type>
    <name>friend[4]</name>
    <description>int_list[4]</description>
   </variable>
   <variable><type>int</type>
    <name>enemy[4]</name>
    <description>int_list[4]</description>
   </variable>
   <variable><type>char</type>
    <name>strategy</name>
    <description></description>
   </variable>
  </memory>
 </xagent>
</agents>
```

Table 5.4: Aggressor-defender game: Defining functions in Flame

```
<functions>
  <function>
   <name>sendinformation</name>
   <currentState>00</currentState>
   <nextState>01</nextState>
   <outputs>
    <output>
     <messageName>agentInformation</messageName>
    </output>
   </outputs>
  </function>
  <function>
   <name>move</name>
   <currentState>01</currentState>
   <nextState>02</nextState>
   <inputs>
    <input>
     <messageName>agentInformation</messageName>
    </input>
   </inputs>
  </function>
</functions>
```

5.2.2 Implementation

Observing the snippets of code provided in Table 5.1, Table 5.2 and Table 5.3, it becomes apparent that NetLogo and Repast have a very similar syntax; which sets them apart from Flame. The NetLogo programming language is fairly simple, a procedural language based on follow up languages of Logo. NetLogo supports external procedures/reporters in Java and Scala, and it has an extension that allows the execution of Prolog inside NetLogo code. The Repast Simphony model was build using ReLogo which is a dialect of Logo. However, Repast Simphony models can be also built by utilizing Groovy or Java. Table 5.1 demonstrates usage of the C programming language for writing functions in Flame. This might represent a problem for the modellers, given that C requires a certain experience and expertise.

Movement functions and manipulation of an agent's position and direction, are the most important spatial characteristics. Unlike Flame, NetLogo and Repast support these characteristics natively. As it can be seen in Table 5.1, Table 5.2 and Table 5.3, the Flame implementation of the coordinates of an agent (the variables *agent_x* and *agent_y*) are a custom user variable of any type chosen by the modeller. This notion makes it more difficult to write Flame models for spatial MAS or to understand existing models. Furthermore, there is an increased level of difficulty when it comes to writing movement functions in Flame. The modeller computes the values of what the new coordinates of the agent should be, and then to update the custom variable that was assigned to hold the values for the agent's coordinates. This is much simpler with NetLogo and Repast. The built in function *facexy* accepts input values for the x and y coordinates of the agent, and automatically updates the coordinates of that agent resulting into movement.

NetLogo and Repast have the advantage of assigning agents to positions randomly, therefore allowing for the set up of different simulation scenarios automatically. Flame requires that agents are individually instantiated in a set up file. This usually involves writing an external program which will help into generation of data for the set up file.

5.2.3 Validation, Verification and Testing

NetLogo, Repast, and Flame support only empirical validation. This technique can be performed by collecting the generated time-series output (or other data) from the simulation and then comparing it to collected data from the real world.

NetLogo does not support verification or testing techniques [95, 94]. Similarly, Repast does not support verification techniques [29]. There is, however, active research towards testing techniques, which include a generic testing framework for agent-based simulation models [15, 29]. Finally, given that Repast is entirely built in Java, it supports a test-driven simulation development [15], i.e. unit tests can be carried out to validate the behaviour of a model.

Although the main unit of Flame is an X-machine which is supported by verification and testing strategies [31, 46], there is not a general methodology to natively employ these techniques into Flame. A possible solution includes verifying and testing an agent's model individually, which would cover only the micro-level of the system. Some research has shown that dependence analysis techniques that aid automated test case generation, can also aid the testing of Flame models [83, 12]. This approach is not natively adopted by Flame.

5.2.4 Visualisation

An interesting concept about the visualization tool in NetLogo and Repast is the fact that they produce output in real time. Flame on the other hand, requires that a model is initially run for a specific number of iterations, producing a textual output. This textual output can be later visualized by utilizing the Flame visualiser [83, 12]. Therefore, the models of NetLogo and Repast are directly executable into visual simulation, while Flame models can be visualized with the use of an external tool.

Comparing the forms of output, Repast can provide a histogram, visual display of the interactions between the agents, or a chart of time-series data. On the other hand, NetLogo supports the following visualization options: line, bar, scatter plots, and visual display of the interactions between the agents as 2D and 3D models. Flame behaves poor in this aspect, i.e. it supports only visual interactions display.

Finally, NetLogo and Repast are characterised with visualisers of a very good quality. On the other hand, the Flame's visualiser is in very early stages of its development.

5.2.5 Supported Size of MAS and HPC

NetLogo supports hundreds or thousands of agents to operate independently, providing a clear picture of the micro-level behaviour of the agents, as well as the macro-level patterns (emergence) within the whole system. One example is the Segregation model, which was inspired from social systems (such as housing patterns in cities) and demonstrates a large-scale patterns model [94]. NetLogo, however, does not support high performance computing.

Repast Simphony is very similar to NetLogo regarding to the supported size of MAS and it also does not support high performance computing. There is another release of this framework, Repast for High Performance Computing (Repast HPC) intended for large-scale distributed computing platforms tested to work in a parallel distributed environment [61]. Repast HPC is written in C++ using Message Passing Interface (MPI) for parallel operations [61].

Flame has a clear advantage by supporting simulations that contain up to millions of agents [17]. Furthermore, only Flame is designed to utilise high performance computing, and has been tested to work on both serial and parallel systems [12].

5.3 Conclusions on the comparison

This chapter provides a survey of visualization platforms that, starting from a formal model, are most suitable for spatial state-based MAS models: NetLogo, Repast and Flame. One of the main purposes of this review is to investigate which simulation platform will be best suited for automated translation of a state-based formal model to executable code for visualisation. The results of the comparison between NetLogo, Repast and Flame, are summarized in Tables 5.5 and 5.6.

It is safe to conclude that NetLogo and Repast possess the features that make them more suitable to code simulation of spatial agents. In contrast, Flame,

Table 5.5: Comparison between NetLogo, Repast and Flame: Modelling and Implementation

Characteristics: NetLogo (NL), Repast Simphony (RS) and Flame (F)	**NL**	**RS**	**F**	**Notes**
Modelling				
State-based modelling	×	√	√	
Other modelling methods	×	√	×	Repast can be built by utilizing point-and-click flowcharts.
Implementation				
Native support of movement functions	√	√	×	
Agents have embedded position and direction	√	√	×	
Agents can be assigned to a random position	√	√	×	
Fairly simple programming language	√	×	×	Repast is more complex than NetLogo. Flame functions are written in C, which requires experience and expertise.
Automatic set up of different simulation scenarios	√	√	×	Flame agents have to be individually instantiated in a set up file.
External language interface	√	√	×	Java, Scala and Prolog for NetLogo. ReLogo, Java and Groovy for Repast.

since it is based on X-machines, inherits XM verification and testing strategies but does not have a built-in functionality for spatial agents. Moreover, Flame supports HPC and models with millions of agents. These advantages make Flame of a particular interest for this research.

Based on the disadvantages of NetLogo, Repast and Flame, there are certain ideas for extending these platforms and developing tools to support spatial MAS better. For instance, Flame can be extended to support spatial properties.

This includes support for modelling movement functions and manipulation of an agent's position and direction natively. Moreover, automatic set up of different simulation scenarios for Flame can be solved by providing a tool which will behave in the NetLogo/Repast manner, with the use of a graphical interface. To support visualization in Flame a tool may be developed to read the produced textual output in real time, and feed the Flame visualiser with partial data as they are being generated. On the other hand, NetLogo, Repast and Flame have very poor native support to verification, validation and testing strategies. If the spatial (formally unverifiable) properties are removed from the spatial MAS, there is still the need to formally verify, validate and test the remaining properties. Therefore, there is a ground for developing tools for automatic translation from a NetLogo, Repast and Flame models, into another platform supported with verification or testing.

5.4 Summary

This chapter covers parts the following objectives:

O2: Identifying illustrative case studies which are scalable (experiments with different numbers of agents), simple to be modelled and have spatial characteristics.

O3: Linkage with simulation platforms in order to observe emergent behaviour.

O4: Proving the appropriateness of the method through simulations and visualisations.

The comparative study of tools for visualisation of state-based spatial multi-agent models was published in [64] and [67].

Table 5.6: Comparison between NetLogo, Repast and Flame: Visualisation, Correctness and HPC

Characteristics: NetLogo (NL), Repast Simphony (RS) and Flame (F)	**NL**	**RS**	**F**	**Notes**
Visualisation				
In real time	√	√	×	
Directly executable as a visual simulation	√	√	×	
Support to other types of display besides visual interactions	√	√	×	NetLogo supports visual interactions display, line, bar, and scatter plots. Repast supports histogram, visual interactions display and a chart of time-series data.
Display quality and easiness to set it up	√	√	×	
Comes as an embedded tool	√	√	×	
Correctness				
Verification support theory	×	×	√	Flame inherits the verification strategies from X-machines.
Validation support theory	√	√	√	They support only empirical validation.
Testing support theory	×	√	√	Unit tests can be carried out to validate the behaviour of a Repast model. Flame inherits the testing strategies from X-machines.
HPC				
HPC support	×	×	√	There is another release for Repast designed for HPC (Repast HPC). Flame HPC works on both serial and parallel systems.
Support of millions of agents	×	×	√	

Chapter 6

Framework for modelling and verification of spatial MAS

Emergence is a pattern appearing in the configuration of the agents, at some instance during the lifetime of the system. Spatial systems exhibit the type of emergence related to the positioning in space (such as line formation, flocks, schools, herds etc.). However, formal verification techniques can only be applied under the assumption that the emergent property is known beforehand. In order to be able to tackle this kind of problems, a research framework, depicted in Fig. 6.1 is proposed. This framework helps in identifying emergent behaviour through the automatic transformation of a formal model to an executable visual simulation [68].

6.1 Overview of the modelling and verification framework

Combining the discussion of modelling, formal verification and simulation, there is a need to identify or create a modelling formalism that will provide:

- Ability to model the internal states of an agent, the agent's perception and the agent's knowledge of the environment.
- Ability to model how the agent changes its internal state and knowledge.

- Ability to represent links and exchange of messages between agents.
- Ability to represent spatial characteristics natively.
- Support to formal verification (ex. model checking).
- Complete testing strategy.
- Support for visual animation tools.

Assuming that we can develop such a formal model, it is interesting to consider what property to check for, that is, whether there is an emergent behaviour a given system. The spatial properties are common to every bio-MAS and they are complex for modelling and verification. An interesting question may arise:

> *How can we handle the spatial properties, as to reduce the complexity in modelling of bio-MAS?*

The proposed framework is depicted in Fig. 6.1. At the top, we start by formal modelling of agents. Such formal models should be able to clearly distinguish modelling of various types of behaviours, such as spatial or other behaviours, communication, dynamic organisation etc. By separating the various behaviours within the same formal model, it is possible to apply different transformations which will facilitate further processing. This can be presented as follows:

- On one hand, the spatial behaviour determined by movement in space, can lead towards visual animation. The latter is a useful informal tool which will help identification (or observation) of potential emergent properties, $properties_1$.
- On the other hand, *suitable abstractions* of spatial behaviour together with the rest of the behaviours can lead towards simulation and logging of time-series data. These could be used to identify patterns of behaviours, $properties_2$.

All of the $properties_1$ and $properties_2$ can be combined and possibly filtered (some properties may be excluded) to produce *a set of desired properties.* Finally, the desired properties (including emergence) can be verified in the original spatial

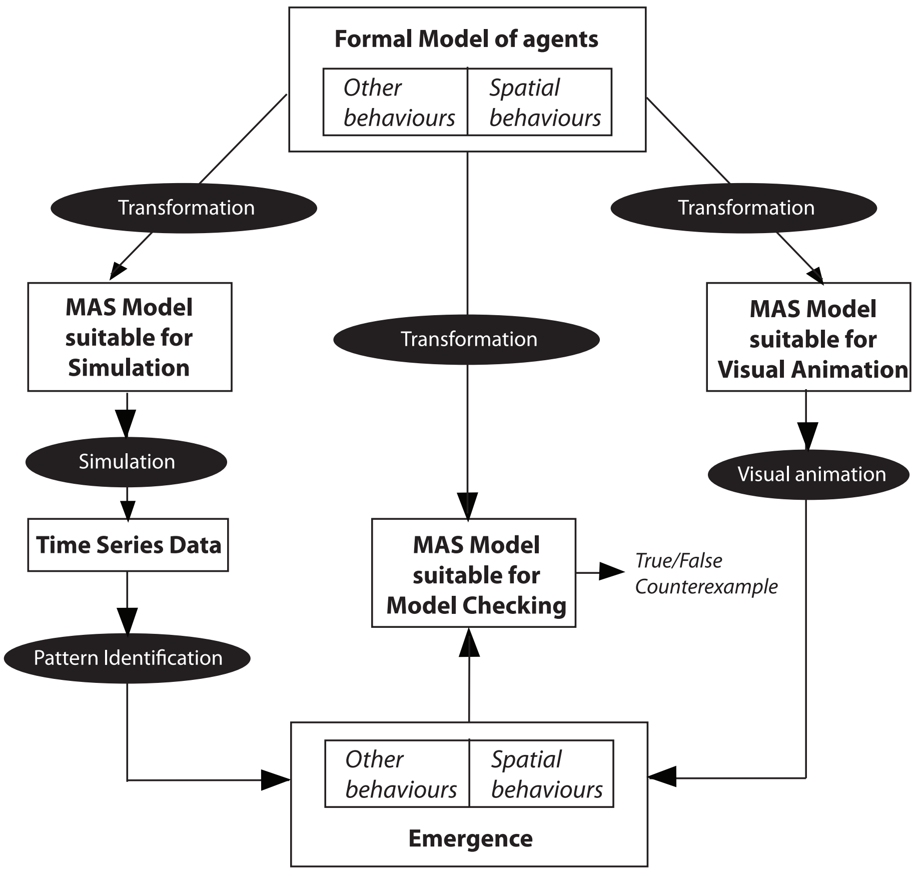

Figure 6.1: A framework for validating emergent properties in spatial biology-inspired MAS.

agent model by model checking, as long as there is a way to transform the original model into an equivalent, susceptible to formal verification, model.

A question may arise:

> *How is it possible to verify the desired properties since it is claimed that verification of spatial properties leads to a state explosion if one tries to model-check a complete model?*

The answer of this question lies in the definition of the model that will be used for formal verification. Basically, this model must contain *suitable abstractions* of the spatial behaviour.

It can be stated that this method might have the following disadvantage:

- If the emergent behaviour is not known in advance, how can one guarantee that it is not introduced by an error in the model?

Therefore, it is desired that the discovered emergent behaviour are carefully examined and the model goes through the all of the steps of the framework possibly more than one time.

6.2 Instantiation of the modelling and verification framework

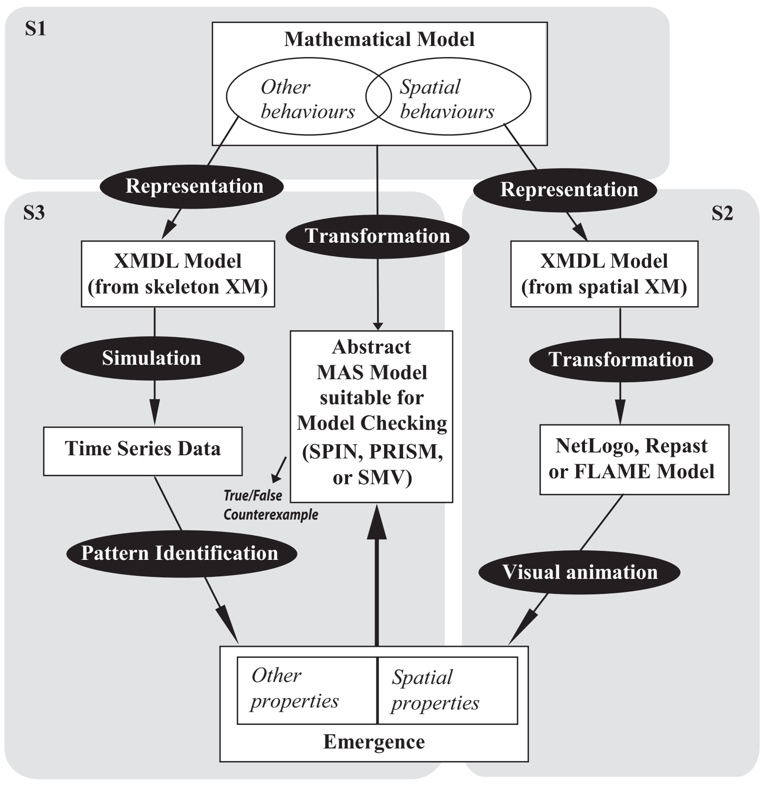

Figure 6.2: A XM instance of the framework for validating emergent properties.

This framework can be instantiated in many different ways. One possibility is to utilise the XMs approach for the initial model, see Fig. 6.2. The spatial behaviour can be detected by utilizing NetLogo. This means that the initial XM model should be translated to an equivalent NetLogo model. The logging of time-series data might be accomplished with a tool such as FLAME. The next step involves utilizing a tool for identifying patterns, such as DAIKON [56]. Finally, the XM can be suitably transformed into an equivalent model in SPIN, PRISM or SMV [33, 50, 55] which will have suitable abstractions of the spatial behaviour. In this case, given a temporal formulae, all of the desired properties could be verified upon the original model. More information about this framework can be found in [68].

It is important to stress that the spatial definition separates the XM memory into two components, which can be observed as:

- A *skeleton XM* that can be used *for verification and testing*; and
- A *spatial XM* that can be used *for simulation and identification of properties.*

To highlight:

- XM model can be translated in code of a simulation tool, which in turn can generate a time-series data.
  - Such tool may be FLAME, which is used to animate XM models with thousands of agents.
  - FLAME does not deal with the spatial behaviour. This can be modelled with NetLogo.
- The logged time-series data can serve as an input to a tool identifying patterns, such as DAIKON.
  - The output would be interesting properties that combined with the emergent properties from visual animation could aid us forming the logic temporal formulae to verify.
- The XM can be suitably transformed into an equivalent model in SPIN, PRISM or SMV with abstractions of the spatial behaviour.
- Given a temporal formulae, it can be verified that all the desired properties hold in the original model.

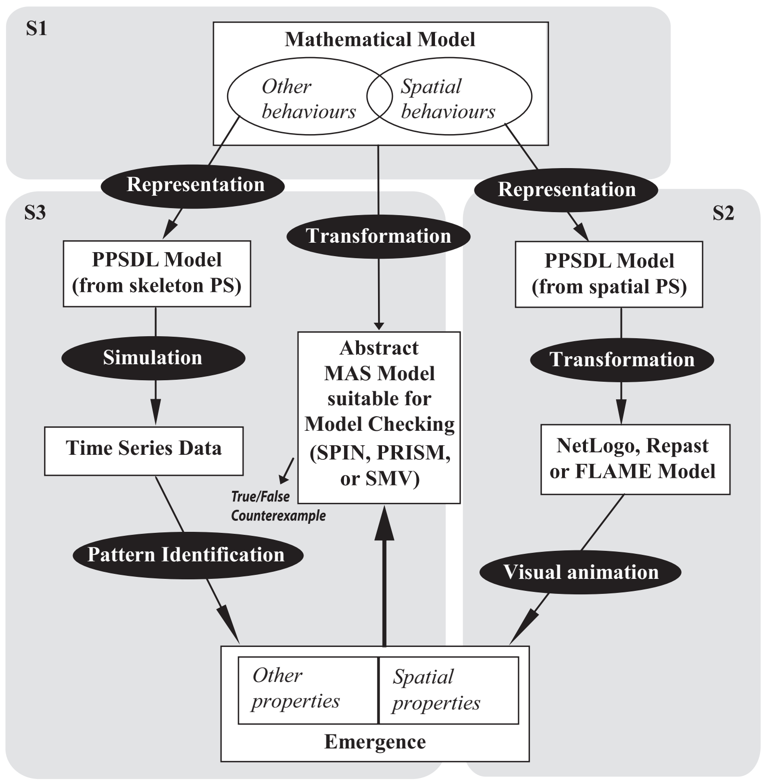

Figure 6.3: A P systems instance of the framework for validating emergent properties.

Other options include utilising P systems approaches for creation of the initial model, Fig. 6.3. Such possibilities could be PPS or kernel P systems (kP systems) [26], which are in essence an unifying framework for P systems. This would be particularly useful because recent research supports verification of various properties of P systems with active membranes and kernel P systems, using different tools like NUSMV [25], SPIN [38] and RODIN [35, 91]. This is very similar to what was already proposed on the XM model as well. The expressive power of kP systems is illustrated in a number of recent investigations [27, 40]. Most interesting for instantiation of the framework is the research that uses the kP system as a modelling tool for biological systems [92], which shows how an X-machine based model developed in FLAME can be naturally transformed into kP system models.

It can be noted that Fig. 6.2 and Fig. 6.3 are divided into three stages: S1, S2 and S3. The following two chapters are going to focus on the stage S1 of Fig. 6.2 and the stage S1 and S2 of Fig. 6.3. Stage S3 in both of them can be considered as future work.

6.3 Summary

The following objective was met with this chapter:

O5: Devising a framework that will combine all of the steps of developing spatial bio-MAS into a process to improve the standard modelling and verification approach for bio-MAS.

The chapter captures a part of the following objective as well:

O4: Proving the appropriateness of the method through simulations and visualisations.

The framework towards the verification of emergent behaviour of spatial MAS was published in [68] and [71].

Chapter 7

Formal modelling of spatial MAS

Biological MAS can be characterised as spatial systems; collections of agents distributed and moving in a physical n-dimensional space. They have incomplete knowledge of the environment and can change their direction and position through time. There are different approaches for modelling spatial phenomena of biological systems, such as:

- Process algebra - can be applied to develop a calculus of processes that could describe the spatial geometric transformations [9].
- Membrane computing - can be utilised by introducing geometric information [82].
- Intracellular NF-κB signalling pathway - an agent-based approach for modelling spatial information in predictive complex biological systems [73].

However, *the combination of biological agents and spatial data modelling* still remains an active research field. For this purpose, modelling spatial properties with the existing modelling formalisms targeted to bio-MAS is considered. The spatial properties are common to every bio-MAS and they are complex for modelling (as presented in Section 2.2). An interesting question may arise:

> *How can we handle the spatial properties, as to reduce the complexity in modelling of bio-MAS?*

The idea utilised involves extending the modelling formalisms. The following sections demonstrate extending XM and PPS in order to support spatial properties. Similar approach is applicable to other formalisms as well. Finally, the model instance in stage S1 of the framework for validating emergent properties in spatial biology-inspired MAS described in 6.2, can be represented with these new structures.

7.1 Spatial XM

Referring back to Figure 2.5, the following shortcomings were identified when modelling spatial agents with XM:

- Even for the simplest model, there might be many different solutions for representing the commonly found properties, such as the initial position or the direction of a spatial agent.
- There are difficulties in simulating a given model because there is not a standard way that deals with the manipulation and processing of the spatial properties like the initial position or the direction of a spatial agent.
- All data structures, including the position and the direction are held in the memory.

Initiated by these shortcomings, a question that can be imposed is: *How can we redefine XMs to support spatial agent modelling natively?* The motivation behind this question can be further broadened into the following aspects:

- The subset of MAS that deals with movement in space is quite numerous, starting with bio-MAS, up to MAS used in many industrial applications like robotics, etc.
- Different modellers might represent a spatial agent's basic characteristics, like position and direction, in different ways.
- The current XM representation for a spatial agent model does not directly map to an animation/simulation.

- The current XM representation for a spatial agent model is rather cumbersome/difficult to code, and in many situations it is also difficult to be understood.
- When it comes to verifying a spatial model with XM, this will result into space explosion due to the spatial information.

7.1.1 Formal definition of spatial XM

There is an element of spatial fit that the biological colonies inhabit, but there is not a straight forward way to formally represent a position, direction or movement utilizing XMs (or any other formalism in this category).

XM are extended by defining three new components (and modifying some existing ones in order to facilitate unification):

- A tuple containing the current position of the agent and an integer that represents its current direction. The current position determines the agent's location in its environment, and the direction represents its heading (such as 30 degrees, 90 degrees, etc.).
- A set containing elementary operations. These operations allow manipulation with the current position tuple and the current direction.

The input and the output set, the memory, the set of states and the next state remain intact, because these structures do not deal with the spatial attributes.

Definition 7.1. The new structure, named *spatial XM (^{sp}XM)* is a 13-tuple ^{sp}XM = (Σ, Γ, Q, q_0, π, π_0, θ, θ_0, M, m_0, E, Φ, F), formally defined as:

- Σ is an input set of symbols;
- Γ is an output sets of symbols;
- Q is a finite set of states;
- q_0 is the initial state;
- M is an n-tuple called memory;

- m_0 is the initial memory;
- π is a tuple of the current position, i.e. $(x,\ y)$ when a 2D representation is considered;
- π_0 is the initial position;
- θ is an integer in the range 0 to 360, that represents a direction (integer values are used as a design choice);
- θ_0 is the initial direction;
- E is a set which contains elementary positioning operations: e_i such as $e_i : \Pi \times \Theta \longrightarrow \Pi \times \Theta$, such as *direction*, *moving forward* and *moving to* a specific position. Here Π and Θ are the sets of π and θ accordingly;
- Φ is a finite set of partial functions ϕ that map a memory state, position, direction and set of inputs to a new memory state, position, direction and set of outputs:
  ϕ: M $\times$ Π $\times$ Θ $\times$ Σ $\longrightarrow$ M $\times$ Π $\times$ Θ $\times$ Γ; and
- F is a function that determines the next state, given a state and a function from the type Φ, such as F:
  Q $\times$ Φ $\rightarrow$ Q.

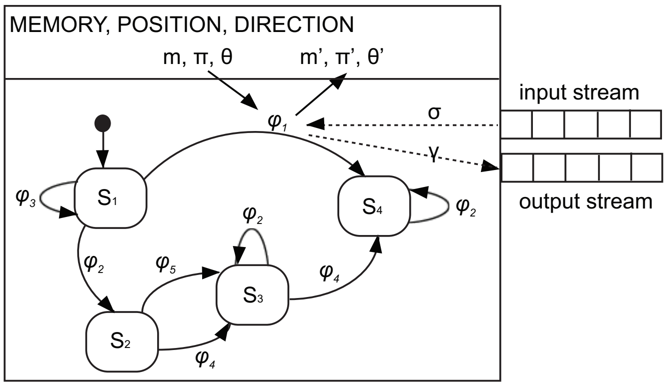

Figure 7.1: An abstract example of a ^{sp}XM.

An abstract example of ^{sp}XM is provided in Fig 7.1. The memory M is composed of M′ ˆ< π, θ >, where M′ is a memory structure from the standard

XM. Moreover, talking about the set which contains elementary operations, there are currently three operations, defined as:

- change_direction m - changes the spatial agent's direction to m, where m is of type θ and m $\in \Theta$ (ex. *change_direction 60*)
- move_n_forward n - moves forward for n units, where n is an integer (ex. *move_n_forward 3*)
- move_to_position x y - moves to specific position (x, y), where x is the x-coordinate, y is the y-coordinate of the agent and (x, y) $\in \Pi$ (ex. *move_to _position 126 43*)

7.1.2 Discussion of ^{sp}XM

The following discussion will concentrate on investigating whether ^{sp}XM inherit the mentioned verification and validation techniques of XMs. An informal proof that an ^{sp}XM is equivalent to any XM could be derived by investigating:

- The memory M of a normal XM is equivalent to the structure of memory M, position Π and direction Θ within an ^{sp}XM. In other words, the position and direction can either become members of the memory tuple in a normal XM model, or they can be excluded from the model without loss of its integrity.
- Any function in an ^{sp}XM model can be translated into a function of the normal XM. More particularly, the predefined spatial operations of a function in an ^{sp}XM model can be omitted or replaced with the standard XMDL syntax to preserve the logic flow.

If a position is found in a precondition of a function it can not be removed because it affect the behaviour. In this situation the position can be kept, but a relatively small set of possible positions may be defined. Alternatively, the position may be translated to a type *Postion*, such as *Postion* = $\{Postion_1$, $Postion_2$ ... $Postion_n\}$, n $\in \mathbb{N}_0$.

Along these lines, by removing the newly defined components that in essence define an ^{sp}XM, what we get is still a completely valid skeleton of a normal

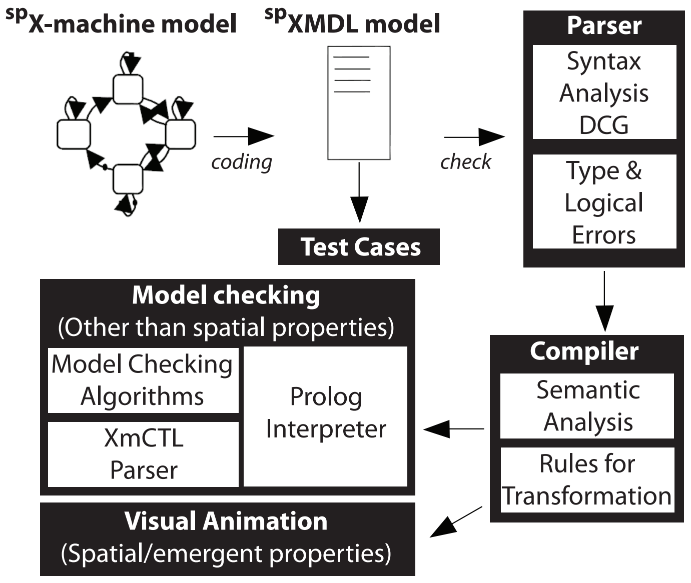

Figure 7.2: Verification and validation of an ^{sp}X-machine.

XM model. Therefore, ^{sp}XMs tend to provide a standardised way of representing some properties of the system, which could also be represented with an XM model and this can lead in easy formalisation, verification (model checking), testing and implementation. The only condition imposed would be not to test or model check the position (coordinates) and direction properties, which in turn will result into state explosion. Finally, ^{sp}XMs are supported by formal verification strategies. They inherit these strategies from XMs. ^{sp}XMDL is facilitated with a parser built using Definite Clause Grammars (DCG) notation [45] as presented in Fig. 7.2.

7.1.3 Modelling with ^{sp}XMDL

A part of the XMDL representation for modelling *Case study 2.1. The foraging ant.* introduced in Sect. 2.4.2 is presented in Table 7.1. The model is based on the new ^{sp}XM approach. In order to support ^{sp}XM representation, the XMDL specification went through some changes. For example, *POSITION* and *DIRECTION* are predefined basic types used to represent the position π or the direction θ of a ^{sp}XM:

```
#position POSITION.
#init_position (2, 3).
```

```
#direction DIRECTION.
#init_direction 60.
```

POSITION and *DIRECTION* can be used for denoting any kind of position or direction, accordingly. As an instance, in the foraging ant case study the memory holds an element of type *POSITION* which in essence represents the position of the seed:

```
#memory (POSITION, seed_id).
#init_memory ((0, 0), nil).
```

The set of operations has the following predefined functions: *change _direction*, *move_x_forward* and *move_to_position*. The operations found in the set of operations could be directly used within a *where* statement of a function in order to instantiate a parameter, This is shown in the function *search_for_base* Table 7.1.

7.2 Spatial PPS

There are many advances in the area of membrane computing up to date; and Population P systems (PPS) with active cells are considered to be one of them [4]. Defined as an arbitrary graph-like membrane structure, they exhibit rather unique characteristics towards modelling multi-agent systems (MAS) with a dynamic configuration (such as biological MAS) [4].

Biological MAS comprised of communities evolving in space and time, such as ant colonies, bacteria populations and skin-like tissues, are characterized with certain dynamicity found not only at the bonding concept, but also at their individual properties. A bonding concept refers to introducing new, or removing old communication links between cells/individuals, which in turn changes during the evolution of a system. On the other hand, individual properties are formed by the notion that a cell can divide or get dissolved; individuals are being born or they die. These two aspects are actually the most prominent behaviour found in this rather complex application's domain, completely supported by PPS with active cells through cell division, cell death and structure reconfiguration, accordingly. However, when it comes to evolution in space, whether it is at individual or community level, it is argued that there is one more piece left out from the

Table 7.1: Modelling the foraging ant case study with ^{sp}XM.

```
#model foraging_ant.

#memory (POSITION, seed_id).
#init_memory ((0, 0), nil).
#position POSITION.
#init_position (2, 3).
#direction DIRECTION.
#init_direction 60.

#fun search_for_seed ((?new_seed_id), (?base_position,
?carrying_seed_id), ?curr_position, ?curr_direction)=
if ?carrying_seed_id = nil and ?new_seed_id = nil then
((ant_keeps_moving_empty), (?base_position, nil),
?new_position, ?new_direction) where
?new_position <- move_x_forward 1 and
?new_direction <- change_direction 10.

#fun search_for_base ((?new_seed_id), (?base_position,
?carrying_seed_id), ?curr_position, ?curr_direction)=
if ?curr_position\= ?base_position and
?carrying_seed_id\= nil then
((ant_searches_for_base), (?base_position,
?carrying_seed_id), ?new_position, ?curr_direction) where
?new_position <- move_to_position ?base_position.
```

puzzle, i.e. the spatial characteristics found in nature. PPS with active cells do not have explicit means and full support to these characteristics, which could be simply defined as the *position*, the *direction* and the *moving function* of a cell or an individual.

7.2.1 Formal definition of spatial PPS

An extension of PPS with active cells, namely *spatial PPS (^{sp}PPS)* introduces the following spatial objects:

- $(\Pi_i{:}\pi_i) \in V$ is a predefined object denoting the cell's position, with values

$\pi_i \in \mathbb{N}_0 \times \mathbb{N}_0$ and a label Π_i that stands for *position*; and

- $(\Delta_i{:}\delta_i) \in V$ is a a predefined object denoting the cell's direction, with values $0 \leq \delta_i \leq 360$ and a label Δ_i that stands for *direction*.

Formally, a cell can be defined as a tuple $C_i = (w_i, \Pi_i, \Delta_i, t_i)$, for each $1 \leq i \leq n$ (where n is the total number of nodes/cells in the system), where $w_i \in V$ is a finite multiset of objects (such as $w_i = (\alpha_i{:}\upsilon_i)$), $t_i \in K$ is a type and the spatial properties (position and direction) are represented as $(\Pi_i{:}\pi_i)$ and $(\Delta_i{:}\delta_i)$.

Other changes that were introduced in the ^{sp}PPS approach are the set of rules $R = R_e \cup \mathrm{R}_s \cup \mathrm{R}_{sp}$, where:

- R_e is the finite set of cell evolution rules (communication, transformation, differentiation, division and death rules) found in the basic PPS with active cells;
- $\mathrm{R}_s = \{((x{:}a);\ (y{:}b),\ in_{copy})_t,\ ((x{:}a);\ (y{:}b);\ enter_{copy})_t,\ ((y{:}b),\ exit_{copy})_t\}$ is the finite set of sensing rules $((x{:}a) \in V,\ (y{:}b) \in V)$; and
- R_{sp} is a finite set of cell spatial rules.

The ^{sp}PPS definition is currently formulated to target spatial movement in 2-D space, however it can be easily generalised for spatial movement in an environment with any dimension n-D. ^{sp}PPS basically define new types of predefined cell objects (that represent the cell's position and direction), new type of communication rules (that define sensing properties) and spatial rules (used to establish movement functions). Complete description and documentation of ^{sp}PPS can be found in [66].

The formal definition can be presented as follows:

Definition 7.2. ${}^{sp}\mathcal{PPS} = (V,\ K,\ C_1,\ C_2,\ldots,\ C_n,\ w_E,\ \gamma,\ \alpha,\ R,\ O)$, such as:

- V is a collection of all the objects from all the cells within the system, including the objects from the environment;
- K is a collection of all the different types, associated with each individual cell in the system to identify different classes/types of cells;

- $C_i = (w_i, \Pi_i, \Delta_i, t_i)$, for each $1 \leq i \leq n$ (where n is the total number of nodes/cells in the system);
- $w_E \in V$ is the multiset of objects initially assigned to the environment;
- γ is the initial structure of the undirected graph, formally defined as:
  $\gamma = (\{1, 2, \ldots n\}, A)$, with $A \subseteq \{\{i, j\} \mid 1 \leq i \neq j \leq n\}$;
- α is a finite set of bond making rules $(t, x_1; x_2, p)$, such as x_1, x_2 $\in V$ and t, $p \in K$;
- $R = R_e \cup \mathrm{R}_s \cup \mathrm{R}_{sp}$;
- O is a partial order over the set of all evolution rules R.

This definition is currently formulated to target spatial movement in 2-D space, however it can be easily generalised for spatial movement in an environment with any dimension n-D. ^{sp}PPS basically defined new types of predefined cell objects (that represent the cell's position and direction), new type of communication rules (that define sensing properties) and spatial rules (used to establish movement functions). In order to define movement and change of direction (to specific or random values), a particular type of rules, called spatial rules, were introduces as follows:

- $(A; \Pi, \textit{move } B)_t$ where $B \in \mathbb{Z} \cup \{\textit{Rand}\}$ – In presence of an object A in a cell of type t, set Π at position B steps forward/backward (depending if B holds a positive or negative value, respectively). If B holds the value *Rand*, set Π at a random number of steps forward/backward;

- $(A; \Pi, \textit{set } B)_t$ where $Y \in \{(\mathbb{N}_0 \times \mathbb{N}_0) \cup \textit{Rand}\}$ – In presence of an object A in a cell of type t, set Π at position B. If B holds the value *Rand*, set Π at a random position;

- $(A; \Delta, \textit{set } B)_t$ where $B = \textit{Rand}$ or $1 \leq B \leq 360$ – In presence of an object A in a cell of type t, set Δ at direction B. If B holds the value *Rand*, set Δ at a random direction;

And finally, for a given subset of evolution or spatial rules r_1, r_2, r_3..., the partial order O would allow ordering, for example r_1, $r_2 \succ r_3$ means that any rule that among r_1 and r_2 can be non-deterministically chosen, as long as they execute before r_3.

This definition is currently formulated to target spatial movement in 2-D space, however it can be easily generalised for spatial movement in an environment with any dimension n-D.

7.2.2 Discussion of ^{sp}PPS

The idea behind ^{sp}PPS is to maintain all the advantages of PPS with active cells, which makes them one of the most suitable formalisms for modelling the macro-level of biologically inspired multi-agent systems with a highly dynamic nature. Some of the most prominent advantages of PPS with active cells are their support towards [87]:

- Non-deterministic communication – As opposed to deterministic communication, this property engages choices from indistinguishable possibilities, i.e., a communication rule is selected randomly;
- Dynamic addition and removal of agent instances – Cells can be introduced in (cell division) or removed from (cell death) the system, during the system's evolution;
- Dynamic restructuring of the communications network – Cells can change the links (bonds) of their interconnection (form new bonds with other cells, or destroy existing bonds) as the system evolves. In other words, this implies to the concept of communication change;
- Maximal and arbitrary parallelism – These properties refer to the support of applying a *maximal/arbitrary* number of rules, but only one instance of any rule (selected non-deterministically) used at each computation step.

^{sp}PPS basically defined new types of predefined cell objects (that represent the cell's position and direction), new type of communication rules (that define sensing properties) and spatial rules (used to establish movement functions). The detailed specifications are going to be further explained, as follows.

All the objects in the system, including the cell's position and direction, as well as the objects that belong to the environment are represented in the form ($label$:$value$), where *label* is a descriptor of the object and *value* holds the actual object's value, such as (age:$\mathbb{N}_0$) or ($temperature$:$\mathbb{R}$). As for the cell's position and direction, they are treated as all the rest of the objects, therefore the existing bond making rules found in the PPS with active cells definition, now allow manipulation with the spatial objects, i.e. (t, (Π_1:π_1); (Π_2:π_2), p) where t, $p \in K$, is a valid bond making rule. Similarly, the spatial objects can be used in all of the evolution rules as well, and there might be objects that belong to the environment of these types (π and δ). ^{sp}PPS however, define the following three new sensing communication rules: $((x{:}a);\ (y{:}b),\ in_{copy})_t$, $((x{:}a);\ (y{:}b),\ enter_{copy})_t$ and $((y{:}b),\ exit_{copy})_t$. The difference from the existing communication rules is that now a copy of an object can be communicated through the cell's membrane, rather than the actual object. If there is already an object within the cell with the same label as the copy, this object will be replaced with the new one. A graphical summary of all the evolution rules that exists in ^{sp}PPS is provided in Fig. 7.3 (types of rules: i.–iii. communication, iv. transformation, v. differentiation, vi. division, vii. death rules and viii.–x. sensing communication rules).

7.2.3 Modelling with ^{sp}PPSDL

Case Study 2.2: Ant lines. introduced in Sect. 2.5.2 can be modelled utilising the new ^{sp}PPS approach as presented in Table 7.2 (note that x represents *any*). This can be compared with the implementation utilizing the PPS approach, already presented in Table 2.4.

On each evolution cycle, all of the rules R are applied in parallel. The new spatial objects defined in ^{sp}PPS are optional and applicable only to the cells that resemble biological moving individuals. On the other hand, an interesting concept introduced within the spatial rules is the awareness of *randomness*, a notion commonly found in the natural biological processes. Finally, the idea behind the new communication rules allows the sensing characteristics, or more

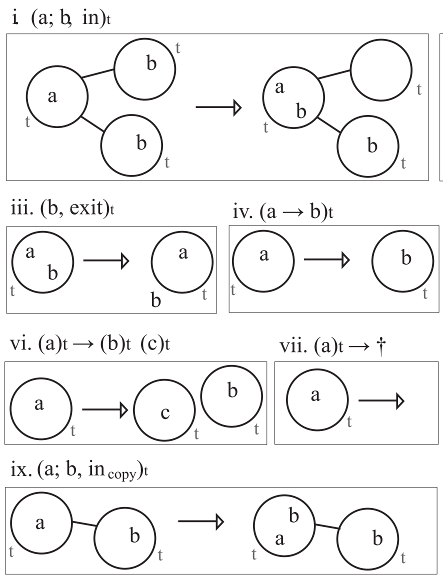

Figure 7.3: Evolution rules in ^{sp}PPS ($a \in V$, $b \in V$).

particularly object multiplication.

7.3 Conclusion

As it can be determined from the definition, a ^{sp}XM in essence provides a separation of the behaviour within the system that deals with the movement (and the other spatial attributes) from the rest of the behaviour. This provides a standardised way to modelling motion, which is easily understandable and provides a direct mapping to an animation/simulation. And finally, this definition maintains an obvious equivalence with the standard XM.

The new definition of ^{sp}PPS expanded the limitations and/or difficulties that were found in this modelling formalism when it comes to spatial agent properties, as discussed in Sect. 2.5. ^{sp}PPS provide an intuitive and flexible way to model spatial and temporal properties. Moreover, it allows modelling of these characteristics with a great level of detail, which in turn would results to a more prominent emergent behaviour. One might argue that *some* of the improvements found in ^{sp}PPS could be actually modelled with the standard PPS approach. However, by introducing all of the new spatial characteristics in the ^{sp}PPS def-

Table 7.2: Modelling the ant lines case study with ^{sp}PPS

$\boldsymbol{V}$ = {($nest\_position$:π),($food\_position$:π), ($timer$:$\mathbb{N}_0$),(ant_no: $\mathbb{N}$)}
$\boldsymbol{K}$ = {$leader\_ant$, $follower\_ant$}
γ = ({ant$_1$, ant$_2$, ant$_3$,... ant$_n$},
{{ant$_1$, ant$_2$}, {ant$_2$, ant$_3$}, ... {ant$_{n-1}$, ant$_n$}})
α = {($leader\_ant$, (ant_no: 1); (ant_no: 2), $follower\_ant$),
($follower\_ant$, (ant_no: i); (ant_no: i + 1), $follower\_ant$)}
$\boldsymbol{w}_E$ = {($nest\_position$: (0, 0)), ($food\_position$: (0, 50))}

$\boldsymbol{C}_1$ = ({(Π_1: $nest\_position$), (Δ_1: 60), (ant_no: 1)}, $leader\_ant$)
$\boldsymbol{C}_i$ = ({(Π_i: $nest\_position$), (Δ_i: 60), (ant_no: i), ($timer$: 0)}, $follower\_ant$), for $2 \le i \le n$

$\boldsymbol{R}$ = {r_1, r_2,...r_6}, such as:
$\boldsymbol{r}_1 = ((timer{:}t) \to (timer{:}\ t + 1))_{follower\_ant}$ – transformation rule, for each of the cells with a type *follower_ant*, increase the timer object by one.

$\boldsymbol{r}_2 = (\Pi_i)_x \to \dagger$, if Π_i = *food_position*, for $1 \le i \le n$ – death rule, once a cell (of any type) reaches at the position of the food, this cell dies.

$\boldsymbol{r}_3 = (x;\ \Delta_1,\ set\ Rand)_{leader\_ant}$ – spatial rule, set random direction to a cell of a type *leader_ant*

$\boldsymbol{r}_4 = (x;\ \Delta_i,\ set\ \Delta_{i-1})_{follower\_ant}$, for $2 \le i \le n$ – spatial rule, set the direction of a cell with a type *follower_ant* to be the set the direction of the preceding cell.

$\boldsymbol{r}_5 = (\mathrm{x};\ \Pi_1,\ move\ Rand)_{leader\_ant}$, $Rand \in$ {-1, 0, 1} – spatial rule, set random position to a cell of a type *leader_ant*

$\boldsymbol{r}_6 = (timer;\ \Pi_i,\ move\ Rand)_{follower\_ant}$, if $timer \ge 2 +$ i where $Rand \in$ {-1, 0, 1}, for $2 \le i \le n$ – spatial rule, set random direction to a cell of a type *leader_ant* if there is a object of type *timer* with value 2 + i.

$\boldsymbol{O} = \{r_1 \succ r_2 \succ r_3,\ r_4 \succ r_5,\ r_6\}$

inition, a greater idea than uniform representation is established, i.e. forming a basis towards informal validation of this modelling formalism.

7.4 Summary

The following objectives were met with this chapter:

O6: Extending the definition of approaches for modelling bio-MAS with geometrical elements into a coherent model.

O7: Extending existing tools with features coming out of the new definitions.

Parts of this research were published in [70], [71] and [66].

Chapter 8

Visualisation and simulation of spatial MAS

8.1 Simulation of ^{sp}XMs

Referring to an earlier discussion for combining formal with informal techniques towards the verification of spatial MAS, it is suggested to utilise visual animation for detecting the emergent properties of a system, such as the NetLogo platform. However, given an XM model, it is not always easy to map its representation into an equivalent NetLogo code. This is due to the already discussed disadvantages in Sec. 7.1 that deal with the behaviour of the system that represents motion (and the other spatial attributes). This raises the question: *Having a model of a system, how can we visualise it?* ^{sp}XMs overcome the problem found in XMs models, and thus enhance visual animation, as the agent's position and direction can be interpreted into motion within an animation platform. This feature opened ideas for automation of the simulation scenarios for an ^{sp}XM model, resulting into a tool ^{sp}XM2Visual.

8.1.1 ^{SP}XM2VISUAL CONVERSION TOOL

The mapping presented in Fig. 8.1 provided inspiration to pursue an automatic mechanism that would translate a ^{sp}XM model into code for the NetLogo platform. NetLogo was chosen as an initial choice and a similar work for Repast is currently in progress. A variation of this idea was presented in [65] concentrating on the semi-automatic translation of XMs to NetLogo code (the user had to manually write NetLogo code for modelling motion in the environment). However, the spatial characteristics of a ^{sp}XM make it possible for a fully automatic translation because now there is a mapping from the ^{sp}XMDL model to the NetLogo functions that represent movement. The tool developed is called ^{sp}XM2Visual, and its components are abstractly presented in Fig. 8.2.

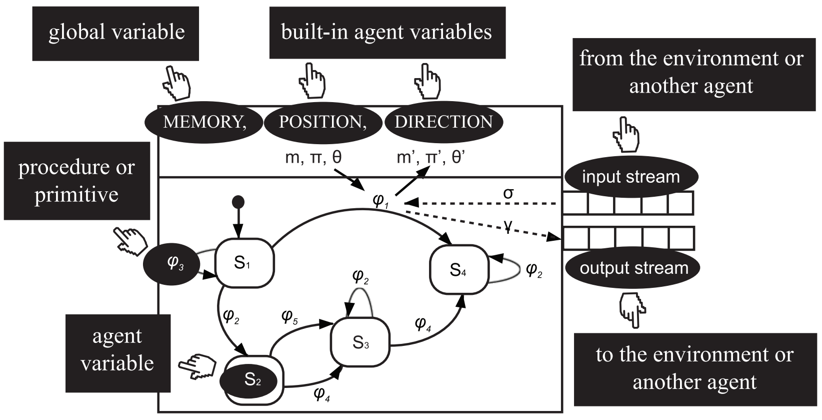

Figure 8.1: ^{sp}XM mapping to NetLogo.

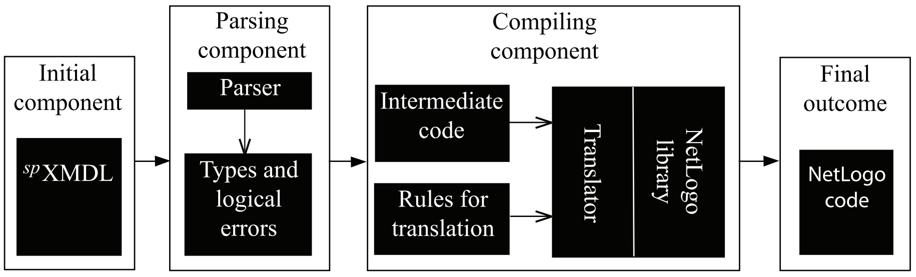

Figure 8.2: System architecture of ^{sp}XM-Visual.

As it can be noticed from Fig. 8.2, the ^{sp}XM2Visual system architecture consists of 2 main components, the parser (reports of possible errors, like types and logical ones) and the compiler (contains all the rules and the logic for the translation). Given that NetLogo supports only lists (this is a mathematical structure similar to the array found in a programming language), in order to produce an equivalent NetLogo model from a ^{sp}XM representation, there was a need of cre-

Table 8.1: Examples of the NetLogo library

Math primitive	**NetLogo structure**	**Operations**	**NetLogo functions**
set	list	$=$	are_sets_equal
e.g. set1={a, b}	e.g.[“a”,“b”]	$\cup$	set_union
set2={a, d, e}	e.g.[“a”,“d”,“e”]	$\cap$	set_intersection

Table 8.2: Rules for transformation

IF	**THEN**
⟨FUNCTION HEAD⟩ where ⟨VAR⟩ ← ⟨EXPR⟩	⟨TRANSLATED FUNCTION HEAD ⟩ `let` ⟨EXPR⟩ ⟨NEW VAR⟩ `where` ⟨NEW VAR⟩ = `?memory_`⟨the position of the element⟩_⟨the name given by the modeller⟩ `if` ⟨NEW VAR⟩ ∈ ⟨MEMORY⟩ or ⟨NEW VAR⟩ = `?input_`⟨the position of the element⟩_⟨the name given by the modeller⟩ `if` ⟨NEW VAR⟩ ∈ ⟨INPUT⟩

ating an external library. This external library for NetLogo (included in the compiler of Fig. 8.2) supports all the mathematical primitives found in ^{sp}XMDL (sets, bags, sequences, etc.) and their operations (see Table 8.1 for examples). Moreover, the set of operations from ^{sp}XMs is translated within this library as functions (an agent's movement to a certain position, the perception of the environment, etc.). Finally, functions that deal with the modelling of the environment (defining obstacles, defining agents, etc.) are included as well.

Except from the library, other interesting parts of the compiling component are: the *rules for transformation* and the *translator*. The rules for transformation are simple *if-then rules*. An example of such a rule for the *where statement* in an ^{sp}XMDL representation is presented in Table 8.2.

The rules for transformation are written as BNF grammar:

- If there is a *where statement* and it contains the sign “←”, then replace this sign with *let* and swap the positions of the operand with the sign.

- If there is an *if statement* and it has a *sign* "=<", then replace it with "<=".

Or more complicated nested *if-then rules*:

- If there is a *where statement* and its condition starts with "?", then replace the variable name with the following formatting α_⟨*the position of the element*⟩_⟨*the name given by the modeller*⟩, such as:
  - If the variable name can be found in the memory tuple, then replace α with *memory*.
  - If the variable name can be found in the set of inputs, then replace α with *input*.
- If there is an *if statement* and it has a *sign* "\\=", then replace it with "!=".
- If there is an *if statement* and it has a *sign* "belongs", then replace it with "member?".
- If there is an *if statement* and it has a *sign* "=<", then replace it with "<=".

On the other hand, the translator is composed of:

- *a reader* - parses a ^{sp}XMDL representation and constructs the necessary objects;
- *an object model* - function, memory, state, transition, etc.; and
- *a writer* - used in writing the NetLogo code.

The translator is coded in Java. For example, Table 8.3 shows the beginning of the *XMachine* class. This class represents an XM and belongs to the *object model* part. Table 8.4 shows the function *constructXM* and this belongs to the *reader* part.

Fig. D.1 in Appendix D shows the class diagram of the translator component. The diagram is divided into 4 parts as explained in Appendix D.

Table 8.3: Beginning of the *XMachine* class of the object model.

```
package objectmodel;

import java.util.ArrayList;

public class XMachine {

public String Name;
public Memory Memory;
ArrayList<Type> Types;
ArrayList<Type> Inputs;
ArrayList<Type> Outputs;
ArrayList<Transition> Transitions;
ArrayList<Function> Functions;

public XMachine() {
    Name = "";
    Memory = new Memory();
    Types = new ArrayList<Type>();
    Outputs = new ArrayList<Type>();
    Inputs = new ArrayList<Type>();
    Transitions = new ArrayList<Transition>();
}
```

Referring back to the framework towards the verification of emergent behaviour of spatial MAS presented in Section 6.1, the XM model can be actually substituted with the new created ^{sp}XM model. The following benefits will be achieved:

- It will be easier to transform a ^{sp}XM model into a simulation tool that can generate a time-series data, such as FLAME.
- There is an automatic translation of ^{sp}XM models to NetLogo with the new ^{sp}XM2Visual tool.

Appendix C presents the compiled code with the ^{sp}XM2Visual tool for a simple implementation of the *foraging ant* case study. It should be noted that the file *xmdl.nls* is the library with mathematical primitives and other functions for modelling.

8.2 Simulation of ^{sp}PPS

^{sp}PPS are supported with formal verification techniques. Spatial multi-agent systems however, are complex in nature. This means that utilizing ^{sp}PPS one

Table 8.4: The *constructXM* method of the *Constructor* class in reader.

```
public void constructXM() {

    Memory memory = new Memory();
    State state = new State();
    ArrayList<Type> xmOutputs = new ArrayList<Type>();
    ArrayList<Type> xmInputs = new ArrayList<Type>();
    ArrayList<Transition> xmTransitions = new
        ArrayList<Transition>();
    ArrayList<Function> xmFunctions = new ArrayList<
        Function>();

    for (int i = 0; i < File.size(); i++) {
        String curr = File.get(i).toString();
        if (curr.startsWith("#model")) {
            constructModel(curr);
        } else if (curr.startsWith("#type")) {
            constructType(curr);
        } else if (curr.startsWith("#memory")) {
            constructMemory(curr, memory);
        } else if (curr.startsWith("#states")) {
            constructStates(curr, state);
        }
    }
    for (int i = 0; i < File.size(); i++) {
        String curr = File.get(i).toString();
        if (curr.startsWith("#init_memory")) {
            constructInitMemory(curr,
                openingAndClosingQuote, memory);
        } else if (curr.startsWith("#init_state")) {
            constructInitState(curr, state);
        } else if (curr.startsWith("#input")) {
            constructInput(curr, xmInputs);
        } else if (curr.startsWith("#output")) {
            constructOutput(curr, xmOutputs);
        } else if (curr.startsWith("#fun")) {
            constructFunctions(curr, xmFunctions,
                memory, xmInputs);
        } else if (curr.startsWith("#transition")) {
            constructTransitions(curr, xmTransitions);
        }

    }
    XMachine.setMemory(memory);
    XMachine.setStates(state);
    XMachine.setInputs(xmInputs);
    XMachine.setOutputs(xmOutputs);
    XMachine.setTransitions(xmTransitions);
    XMachine.setFunctions(xmFunctions);
}
```

would need an exponential time to complete the execution of a model checker, because a thorough exploration on the system's state means all possible positions (coordinates) and directions.

Along these lines, PPS are supported with testing strategies as well. Testing is a dynamic verification technique by providing all possible series of inputs and comparing its outputs with the documented specification [32]. Therefore, the same concept from this discussion (an exponential time to complete all the test) would apply for the idea of testing ^{sp}PPS models. This leaves simulation as the only appropriate way to confirm that the PPS model is having the intended behaviour. Simulation refers to executing scenarios (animations) and comparing the expected behaviour of the system to a textual or visual (visual animation) outcome. We claim that informal verification (or more specifically, simulation in a form of visual animation) in biologically inspired, spatial agent modelling provides the following benefits:

- *Benefit 1:* Detecting an emergent behaviour that cannot (or it is rather cumbersome to) be realised from a formal representation.
- *Benefit 2:* Visual ensurance that the behaviour of the system or an individual within the system, acts according to the intended specification/requirements.
- *Benefit 3:* Can act as a white box testing mechanism to systems that are NP-hard (their output cannot be verified in polynomial time) or in some cases, even NP-complete.

These concepts are going to be supported by simulating the *Ant Lines* case study, introduced in Sect. 2.5.2. For this purpose, NetLogo [95], [94] as a programmable platform for visual animation of multi-agent systems, is going to be utilised. The main reasons for choosing this software tool are: the support of large-scale systems, the utilisation of a functional language as a back-end (which in turn is perfect for representing an agent's behaviour) and finally, the customisable graphical interface that facilitates modelling of the environment. The NetLogo simulation of this case study is presented in Fig. 8.3 (this simulation comes with the NetLogo's built-in library).

For the purpose of investigating the trace of the ants' path, Fig. 8.3 a) shows a mark being formed as the *leader ant* moves. Followed by the *follower ants*, it will be noticed that the shape of the ant line changes over time. Finally, a trace is left

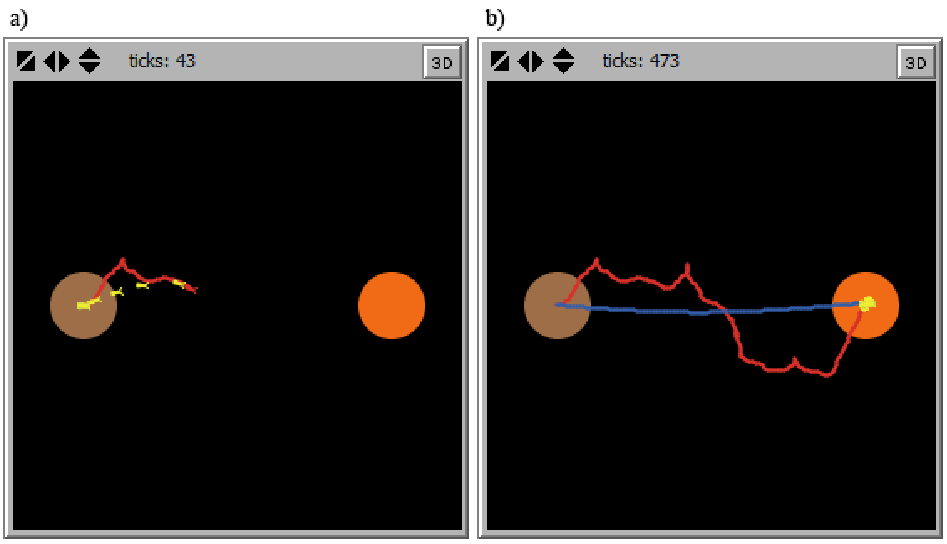

Figure 8.3: NetLogo simulation of the Ant Lines example.

by the last ant, as shown in Fig. 8.3 b). Interesting observation at this point is the comparison of the two paths left from the initial and final ants; the path from the leader is rather circuitous in comparison to the smooth shaped trail formed by the last ant. This simulation clearly demonstrates an emergent behavior that could not be detected from the dry PPS or ^{sp}PPS formal representation, i.e. *Benefit 1*. Furthermore, by observing the shape of the ant line as it is being formed, it can be ensured that the ants really follow the direction of the ant in front; which also ensures that the system acts according to the intended requirements as expected in *Benefit 2*. Finally, regarding the concept presented as *Benefit 3*, we bring into notice that NetLogo is a rather interactive environment, allowing for run-time user inputs and change of the global parameters.

The new modifications introduced with this work allow a direct mapping between the ^{sp}PPS formal notation to a NetLogo code. For instance:

- `(nest-x nest-y)` and `(food-x food-y)` are the NetLogo definitions for the location of centre of nest and location of centre of food, supported in ^{sp}PPS by predefined object types (i.e. the position $\pi \in \mathbb{N}_0 \times \mathbb{N}_0$);

- `leader-heading` is representation of the heading of a leader ant, supported in ^{sp}PPS by the predefined object Δ denoting the cell's direction;

- `set leader-heading [heading]` are net logo functions, now directly supported with the spatial rules; this specific function matches $(X;\, \Delta,\ set\ Y)_t$;
- `rt random-float angle` is the NetLogo function that produces a random number (in this particular case, a floating point number) and assigns it to the direction of the ant. ^{sp}PPS can represent this behaviour by the notion of randomness introduced in the spatial rules;
- `breed [leaders leader]` and `breed [followers follower]` are supported by defining cell types.

There are many other similar examples that could demonstrate the logical link between NetLogo and ^{sp}PPS. This lead to the idea that by defining some common rules for transformation between a ^{sp}PPS model and a NetLogo code, it is now possible to develop a tool that would do this translation automatically. This future direction will actually enhance ^{sp}PPS with informal verification strategies.

8.3 Summary

The following objectives were met with this chapter:

O3: Linkage with simulation platforms in order to observe emergent behaviour.

O4: Proving the appropriateness of the method through simulations and visualisations.

O7: Extending existing tools with features coming out of the new definitions.

Parts of this research were published in [70], [71] and [66].

Chapter 9

Discussion and evaluation

9.1 Summary

The subset of MAS that deal with movement in space is quite large. It includes complex bio-MAS such as: collective robotics, swarms of unmanned air vehicles (UAVs) or underwater vehicles (UUVs), insect and animal societies, or even human behaviour [2]. Formal modelling and verification of spatial agents is a very complex task. On one hand stands the fact that the verification process leads to combinatorial explosion, because modelling these agents means modelling of their spatial properties (such as position or direction). Therefore, the verification would require exploration of a state space developed by the combination of all agent positions evolved through time [68]. On the other hand, there is the fact that the emergent properties of the system should be known in advance in order to be verified. The concept of emergence can be explained as a pattern appearing in the configuration of the agents, at some instance during the lifetime of the system. In biology or biology-inspired agents the emergence can be observed in-vivo (for example, line formation, flocks, schools, herds etc.). However, when it comes to artificial agents, it is not always straightforward. Driven from these two problems, it might be desirable to combine several formal with informal techniques that would be able to join forces towards the verification of spatial MAS [68].

The support to formal verification of XM, PPS, OPERAS and FLAME, can be summarized as follows:

- X-machines are accompanied by a model checking technique (where $XmCTL$ is used to express logic formulae [46]) and complete testing (under certain assumptions, using a method derived from Chow's W-method for Finite State Machines [34, 20]).
- Verification still remains problematic when Communicating X-machines are considered. The only possible approach is to formally verify and test individual components, but not the system as a whole.
- P Systems and PPS are supported with formal verification and test techniques as well.
- The OPERAS framework always carries the legacy of the formal methods used in each OPERAS component.
- Although the main unit of FLAME is an X-machine which is supported by verification and testing strategies, there is not a general methodology to natively employ these techniques into FLAME. A possible solution includes verifying and testing an agent's model individually, which would cover only the micro-level of the system.

9.2 Contribution

Some of the problems in developing spatial bio-MAS are:

- There are difficulties in simulating a given model because there is not a standard way that deals with manipulation and processing of the spatial properties (the initial position or the direction of a spatial agent).
- The representation of spatial model does not directly map to an animation/simulation.
- The representation of spatial model is rather cumbersome/difficult to code, and in many situations it is also difficult to be understood.
- When it comes to verifying a spatial model, this will result into space explosion due to the spatial information.

Therefore, a hybrid approach of utilizing visual animation as an informal verification technique in parallel with formal verification (where applicable) was proposed.

The aim behind this work is:

The definition of an abstract model supporting elementary geometry, the development of a methodology to build agent based systems using this concept that allows the simulation and visual representation of the systems.

This aim was accomplished by developing a framework for modelling spatial MAS, which helps in identifying emergent behaviour through the automatic transformation of a formal model to an executable visual simulation. This framework facilitates simulation and visual representation of systems.

The objectives were accomplished as presented in Table 9.1. These objectives were achieved as follows:

- **O1** - Chapter 2 talked about modelling bio-inspired systems as spatial MAS, Chapter 3 presented different tools for modelling these systems, Chapter 4 introduced formal verification, simulation and validation concepts for bio-systems, and Chapter 5 provided introduction of several visual simulation platforms and comparison of these simulation platforms based on the several criteria;

- **O2** - Chapter 2 and Chapter 4 presented the following illustrative case studies: Case Study 2.1. *The foraging ant*, Case Study 2.2. *Ant lines* and Case Study 4.1. *Aggressor-Defender*;

- **O3** - Chapter 5 and Chapter 8 talked about visual simulation platforms, visualisation and simulation of spatial MAS. Observable emergent behaviour was presented through case studies;

- **O4** - Chapter 5 and Chapter 8 discuss visualisation and simulation as strategies for proving the appropriateness of the method (by utilising newly introduced modelling formalisms);

- **O5** - Chapter 6 introduces a framework for modelling and verification of spatial MAS;
- **O6** - Chapter 7 provides definitions for extending two modelling formalisms, in order to support spatial properties; and
- **O7** - Chapter 7 and Chapter 8 present a tool developed and discuss towards translation of a system modelled with a finite state machine modelling approach into executable code of a visual simulation platform.

Table 9.1: Mapping of the objectives with the corresponding chapters

Objective	*Chapters*
O1: Investigate on spatial systems, modelling formalisms for spatial bio-MAS outlining properties and disadvantages of existing modelling formalisms, as well as verification and simulation strategies and how can they be enhanced to better support complex spatial bio-MAS.	Chapter 2, Chapter 3, Chapter 4 and Chapter 5.
O2: Identifying illustrative case studies which are scalable (experiments with different numbers of agents), simple to be modelled and have spatial characteristics.	Chapter 2 and Chapter 4.
O3: Linkage with simulation platforms in order to observe emergent behaviour.	Chapter 5 and Chapter 8.
O4: Proving the appropriateness of the method through simulations and visualisations.	Chapter 5 and Chapter 8.
O5: Devising a framework that will combine all of the steps of developing spatial bio-MAS into a process to improve the standard modelling and verification approach for bio-MAS.	Chapter 6.
O6: Extending the definition of approaches for modelling bio-MAS with geometrical elements into a coherent model.	Chapter 7.
O7: Extending existing tools with features coming out of the new definitions.	Chapter 7 and Chapter 8.

9.3 Evaluation and future work

The problems found in XM and PPS (stated in Sec. 4.4, Sec. 7.1 and Sec. 9.2) can be overcome by using their spatial extensions, ^{sp}XM and ^{sp}PPS. They do not only facilitate the process of modelling, but the process of verification as well. ^{sp}XM and ^{sp}PPS, provide consistent means and full support on the spatial characteristics found in nature and expand the limitations and/or difficulties that were found in original modelling formalisms. They provide an intuitive and flexible way to model spatial and temporal properties. Moreover, it allows their modelling with a great level of detail, which in turn *results to a more prominent emergent behaviour*.

Besides the theoretical contribution that was discussed, a focus on the practical benefits was raised as well. Explicitly, there is now a direct mapping between the ^{sp}XM and ^{sp}PPS formal notation to a the NetLogo (or similar) platform for visual simulation. This might be expanded with the following future work:

- Defining rules for transformation between ^{sp}PPS and NetLogo, similar to the ones defined for XMs. This would lead into developing a tool for semi-automatic translation of ^{sp}PPS models to NetLogo executable code.
- Developing translation tools of ^{sp}XM and ^{sp}PPS models into executable code for other simulation platforms, such as Repast or FLAME.
- Providing a spatial extension to the FLAME framework along the same lines of ^{sp}XM and ^{sp}PPS.

There are quite a few other ideas for future work on the concept of improving ^{sp}PPS. They can be summarized as follows:

- Defining another type of rules that will deal with manipulation of the objects within the cells. This is very similar to what was achieved with introducing the sensing rules, but it can be further extended.
- The concept of ordering rules raises a question: *How could dynamic ordering of rules be modelled?* This means that at every cycle, the order by which the evolution rules execute should be recomputed.

- The new spatial support is very simplistic in order to support the basic principles of PPS, but this notion can be extended much further.

Formal verification accompanied with simulation as an informal verification technique, would help into discovering flaws of the formally unverifiable dynamic communication within a bio-system. Moreover, it would provide means to facilitate the communication gap between the formal experts and the biologists (which in turn have no formal background) by providing an immediate feedback understandable to both of the teams. This whole process is further highlighted with the supporting framework to modelling and verification of bio-MAS. Future work includes the stage S3 of the framework, as presented in Fig. 6.2 and Fig. 6.3, namely:

- Automatic transformation of ^{sp}XM or ^{sp}PPS into a simulation tool that can generate a time-series data, such as FLAME.
- A process to utilize a tool for identifying patterns (such as DAIKON) from the logged time-series data.
- A process to form logic temporal formulae from the identified patterns.
- Automatic transformation of ^{sp}XM or ^{sp}PPS into an equivalent model in SPIN, PRISM or SMV.

Appendix A: List of Author's Publications

Short description on the publications up to date, is presented as follows in chronological order:

1. *I. Petreska and P. Kefalas. Towards Novel Approaches to Modelling and Verification of Biologically Inspired Multi-Agent Systems.* - In this work we focus on NetLogo and an abstract architecture of a system that could fully transform MAS models to NetLogo is presented. In principle, this paper's main objectives are to set the foundations of future developments in this area and to pose all the interesting research questions that arise. *Presented in the 5th Annual SEERC Doctoral Student Conference (DSC 2010), 2010* [65].

2. *I. Petreska, P. Kefalas and M. Gheorghe. A Framework towards the Verification of Emergent Properties in Spatial Multi-Agent Systems.* - In this paper we present a framework of how formal modelling can lead towards identification and verification of emergent properties of spatial biology-inspired MAS. We discuss the problem in question as well as initial work done on the formal modelling side and the visual animation of these formal models. *Presented in the Workshop on Applications of Software Agents (WASA 2011), 2011* [68].

3. *I. Petreska and P. Kefalas. Population P Systems with Moving Active Cells.* - In this work we introduce a class of population P systems with moving active cells, namely ^{sp}PPS. We argue that the spatial properties might

lead to more accurate emergent behaviour on the macro-level of a multi-agent system. Moreover, we demonstrate how these properties might bring ^{sp}PPS closer to informal verification strategies and enhance their models with visual animation. *Presented in the Twelfth International Conference on Membrane Computing (CMC12), 2011* [66].

4. *I. Petreska, P. Kefalas and M. Gheorghe. Informal Verification by Visualisation of State-based Formal Models.* - This paper introduces a mechanism for semi-automatic transformation of a spatial X-machine agent models into executable code for a visual simulation environment, namely NetLogo. The rules governing the transformations and the mapping between constructs are described as well. We discuss the implementation and present an example of how visualisation rather than formal versification could assist the understanding of emergent properties. *Presented in the 6th Annual SEERC Doctoral Student Conference (DSC 2011), 2011* [69].

5. *I. Petreska, P. Kefalas, M. Gheorghe and I. Stamatopoulou. Extending X-machines to Support Representation of Spatial 2-D Agents.* - Starting with the notion of modelling biologically inspired agents, this paper focuses on their spatial characteristics. This approach resulted into a novel progression, Spatial X-machines, without retracting the legacy characteristics of X-machines such as testing and verification strategies. *Presented in the 4th International Conference on Agents and Artificial Intelligence (ICAART 2012), 2012* [70].

6. *I. Petreska. Tools for Visual Simulation of MAS Models.* - This work examines three different simulation platforms (NetLogo, Repast and FLAME) under various factors, such as: modelling, design, implementation, verification, testing, visualisation and others. The results of the comparison shows their advantages and disadvantages when it comes to simulating multi-agent systems with spatial characteristics. *Presented in the 7th Annual SEERC Doctoral Student Conference (DSC 2012), 2012* [64].

7. *I. Petreska, P. Kefalas, M. Gheorghe and I. Stamatopoulou. ^{sp}X-Machines: Formal State-Based Modelling of Spatial Agents.* - An extended work of

Spatial X-machines was published for *Communications in Computer and Information Science* series of Springer Berlin Heidelberg [71].

8. *I. Petreska and I. Stamatopoulou. A comparative study of tools for visualisation of state-based spatial multi-agent models.* - A comparative study of tools for visualisation of state-based spatial multi-agent models, as an extended work of [64], was *published for the Proceedings of the 2013 Balkan Conference in Informatics (BCI'13), 2013* [67].

The order of importance is as follows: 7, 5, 3, 8, 1, 4, 6, 2.

Appendix B: The *Aggressor-Defender* case study in NetLogo.

```
turtles-own [
  friend
  enemy
]

to setup
  clear-all
  create-turtles number-of-turtles
  ask turtles [
    set xcor random-xcor
    set ycor random-ycor
    set color one-of [ red blue ]
    set friend one-of other turtles
    set enemy one-of other turtles
  ]
end

to go
  ask turtles
  [
    if strategy = "Defend"
    [
      defend
      ]
    if strategy = "Flee"
    [
      flee
      ]
    if strategy = "Some defend some flee"
    [
      if (color = red)
      [
        defend
        ]
      if (color = blue)
      [
        flee
        ]
      ]
    fd 0.1
  ]
  tick
end

to defend
  facexy ([xcor] of friend + [xcor] of enemy) / 2
         ([ycor] of friend + [ycor] of enemy) / 2
end

to flee
  facexy [xcor] of friend + ([xcor] of friend - [xcor] of
     enemy) / 2
```

```
        [ycor] of friend + ([ycor] of friend - [ycor] of
          enemy) / 2
end

;;;DefaultNetLogoCode
```

Appendix C: Compiled code with the ^{sp}XM2Visual tool for the *foraging ant* case study.

```
__includes ["xmdl.nls"]

to setup
        set_model ["foraging_ant" [[2 3] [0 0] ""] ["
           carrying_nothing"]]
clear-all
create-turtles 1
        ask turtles [run_model "foraging_ant"]
end


to run_model [model_name]
while [true]
[
        let input_0_position user-input "Enter the input
           position: "
        let input_1_seed_id user-input "Enter the input
           seed_id: "
```

```
if get_curr_state model_name = ["carrying_nothing
   "] and
search_and_see_seed model_name input_0_position
   input_1_seed_id = true
[
        set_curr_state model_name ["
           carrying_nothing"]
]
if get_curr_state model_name = ["carrying_nothing
   "] and
search_for_seed model_name input_0_position
   input_1_seed_id = true
[
        set_curr_state model_name ["
           carrying_nothing"]
]
if get_curr_state model_name = ["carrying_seed"]
   and
search_for_base model_name input_0_position
   input_1_seed_id = true
[
        set_curr_state model_name ["carrying_seed
           "]
]
if get_curr_state model_name = ["carrying_seed"]
   and
leave_seed_at_base model_name input_0_position
   input_1_seed_id = true
[
        set_curr_state model_name ["carrying_seed
           "]
]
```

```
]
end


to-report search_and_see_seed [model_name input_0_position
    input_1_seed_id]

        let curr_memory get_curr_memory model_name
        let memory_0_position item 0 curr_memory
        let memory_1_position item 1 curr_memory
        let memory_2_seed_id item 2 curr_memory

        if memory_0_position != memory_1_position and
           memory_0_position != input_0_position and
           memory_2_seed_id = "" and input_1_seed_id != ""
        [
                output-print "ant_detected_and_picked_seed
                   "
                let memory []
                set memory lput input_0_position memory
                set memory lput memory_1_position memory
                set memory lput input_1_seed_id memory

                set_curr_memory model_name memory
                report true
        ]
        report false
end


to-report search_for_seed [model_name input_0_position
   input_1_seed_id]
```

```
        let curr_memory get_curr_memory model_name
        let memory_0_position item 0 curr_memory
        let memory_1_position item 1 curr_memory
        let memory_2_seed_id item 2 curr_memory

        if memory_0_position != memory_1_position and
           memory_0_position != input_0_position and
           memory_2_seed_id = "" and input_1_seed_id = ""
        [
                output-print "ant_keeps_moving_empty"
                let memory []
                set memory lput input_0_position memory
                set memory lput memory_1_position memory
                set memory lput memory_2_seed_id memory

                set_curr_memory model_name memory
                report true
        ]
        report false
end


to-report search_for_base [model_name input_0_position
   input_1_seed_id]

        let curr_memory get_curr_memory model_name
        let memory_0_position item 0 curr_memory
        let memory_1_position item 1 curr_memory
        let memory_2_seed_id item 2 curr_memory

        if memory_0_position != memory_1_position and
           memory_0_position != input_0_position and
           memory_2_seed_id != ""
        [
                output-print "ant_searches_for_base"
                let memory []
                set memory lput input_0_position memory
                set memory lput memory_1_position memory
                set memory lput memory_2_seed_id memory

                set_curr_memory model_name memory
                report true
        ]
        report false
end


to-report leave_seed_at_base [model_name input_0_position
   input_1_seed_id]

        let curr_memory get_curr_memory model_name
        let memory_0_position item 0 curr_memory
        let memory_1_position item 1 curr_memory
        let memory_2_seed_id item 2 curr_memory

        if memory_0_position = memory_1_position and
           memory_0_position != input_0_position and
           memory_2_seed_id != ""
        [
                output-print "ant_found_base_and_left_seed
                   "
                let memory []
                set memory lput input_0_position memory
                set memory lput memory_1_position memory
                set memory lput "" memory
                set_curr_memory model_name memory
                report true
        ]
        report false
end


;;;DefaultNetLogoCode
```

Appendix D: Class diagram of the translator component of the ^{sp}XM2Visual tool.

The class diagram of the translator component is presented in four parts (due to space restrictions) as illustrated in Fig. D.1.

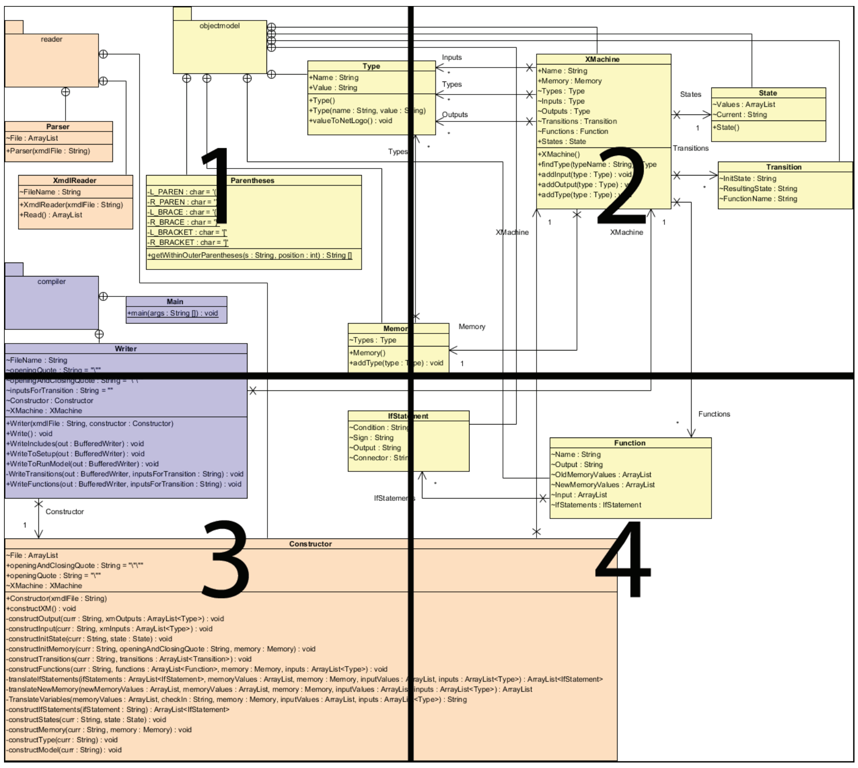

Figure D.1: The class diagram of the translator component.

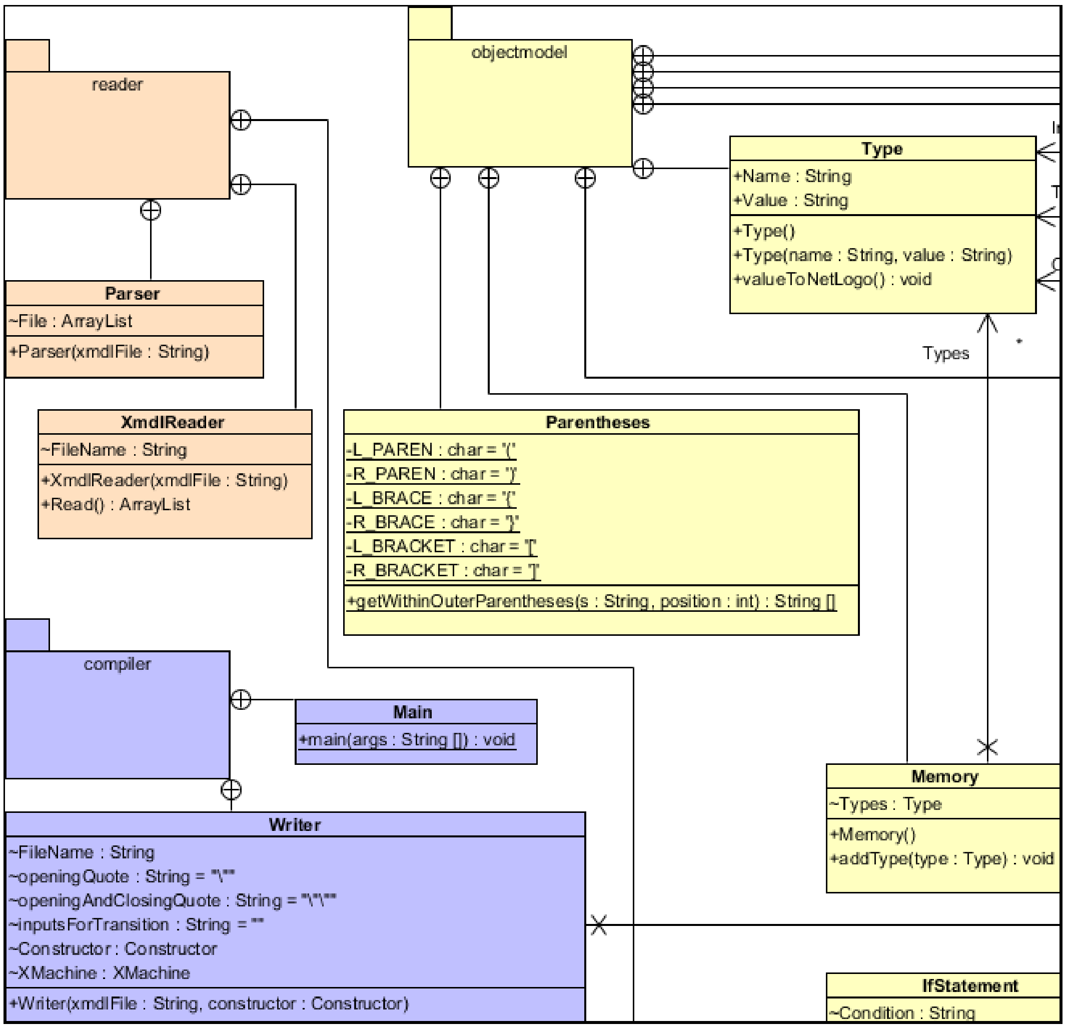

Figure D.2: The translator component of the ^{sp}XM2Visual tool, part 1.

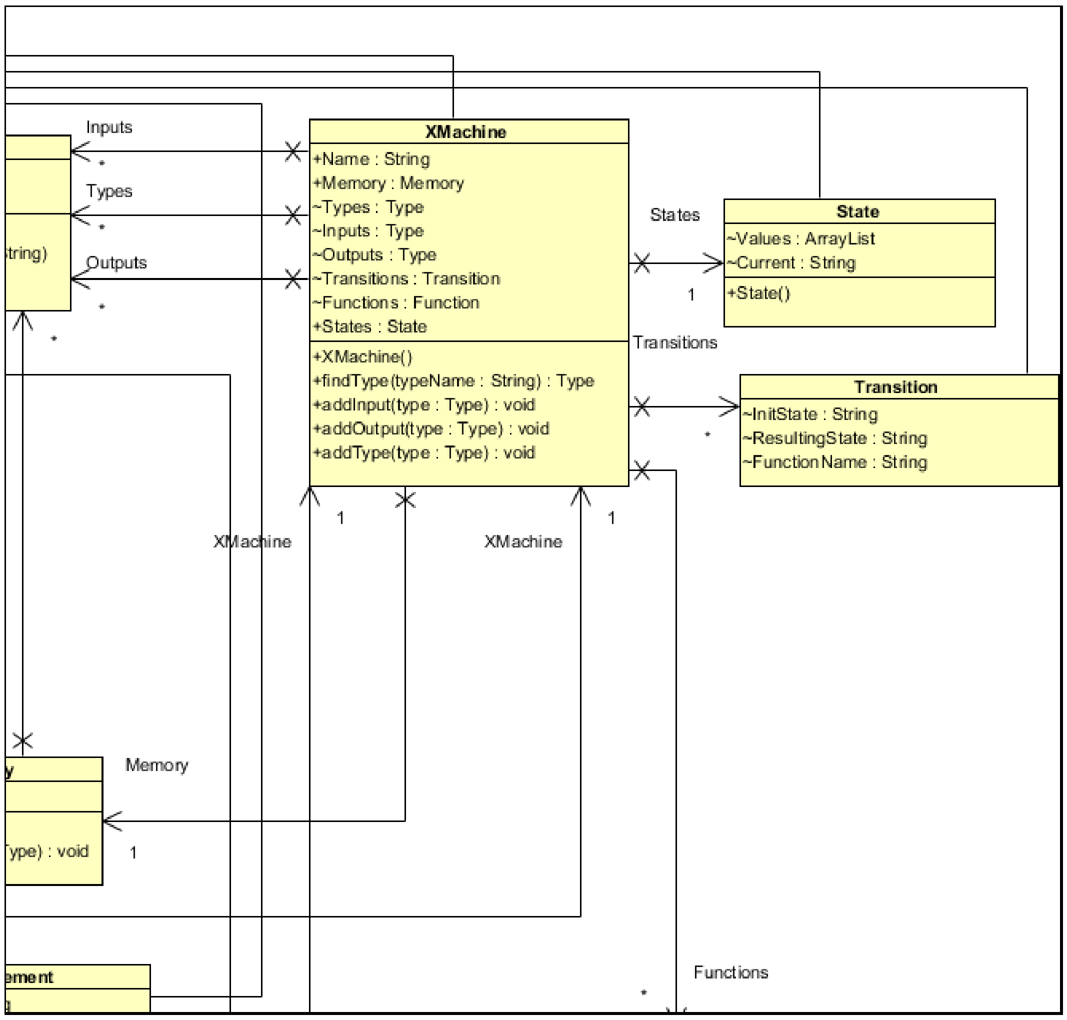

Figure D.3: The translator component of the ^{sp}XM2Visual tool, part 2.

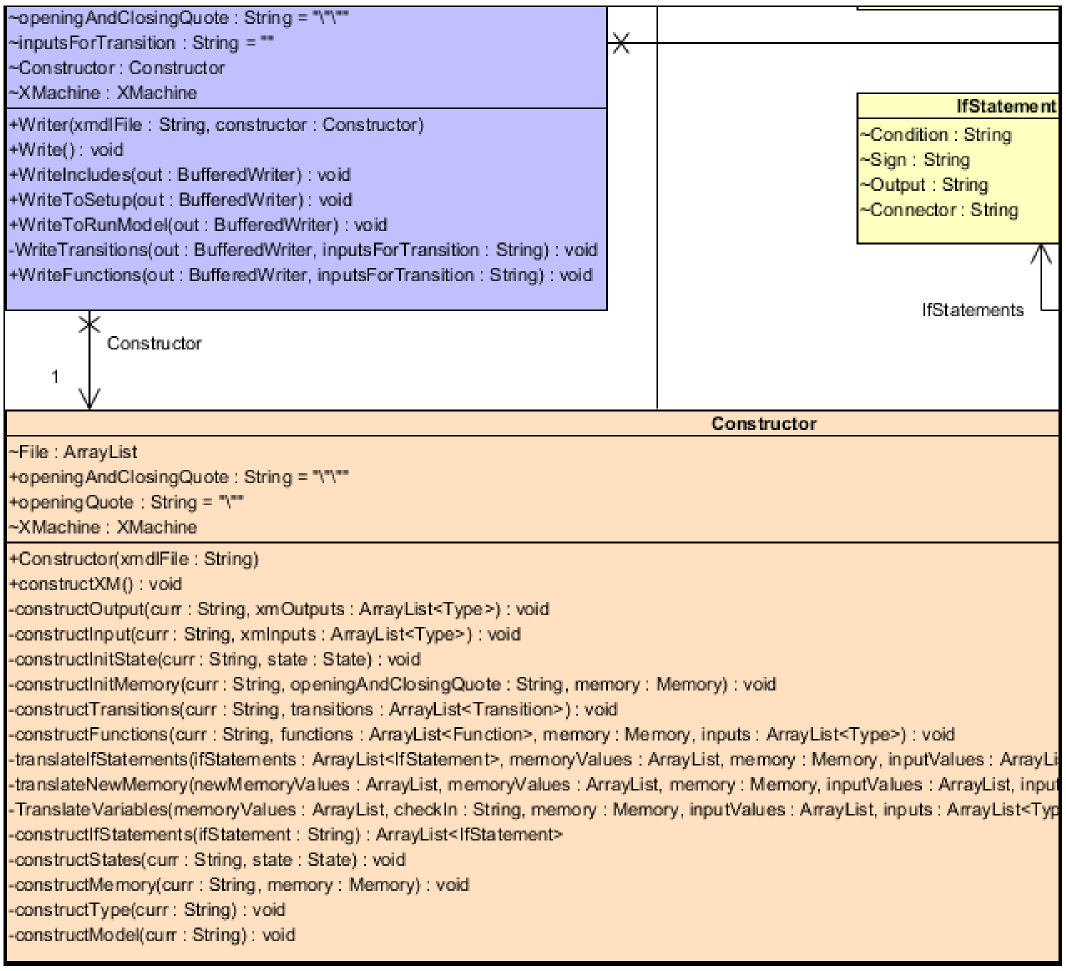
~openingAndClosingQuote : String = "\"\""
~inputsForTransition : String = ""
~Constructor : Constructor
~XMachine : XMachine
+Writer(xmdlFile : String, constructor : Constructor)
+Write() : void
+WriteIncludes(out : BufferedWriter) : void
+WriteToSetup(out : BufferedWriter) : void
+WriteToRunModel(out : BufferedWriter) : void
-WriteTransitions(out : BufferedWriter, inputsForTransition : String) : void
+WriteFunctions(out : BufferedWriter, inputsForTransition : String) : void
IfStatement
~Condition : String
~Sign : String
~Output : String
~Connector : String
IfStatements
Constructor
1
Constructor
~File : ArrayList
+openingAndClosingQuote : String = "\"\""
+openingQuote : String = "\""
~XMachine : XMachine
+Constructor(xmdlFile : String)
+constructXM() : void
-constructOutput(curr : String, xmOutputs : ArrayList<Type>) : void
-constructInput(curr : String, xmInputs : ArrayList<Type>) : void
-constructInitState(curr : String, state : State) : void
-constructInitMemory(curr : String, openingAndClosingQuote : String, memory : Memory) : void
-constructTransitions(curr : String, transitions : ArrayList<Transition>) : void
-constructFunctions(curr : String, functions : ArrayList<Function>, memory : Memory, inputs : ArrayList<Type>) : void
-translateIfStatements(ifStatements : ArrayList<IfStatement>, memoryValues : ArrayList, memory : Memory, inputValues : ArrayLi
-translateNewMemory(newMemoryValues : ArrayList, memoryValues : ArrayList, memory : Memory, inputValues : ArrayList, input
-TranslateVariables(memoryValues : ArrayList, checkIn : String, memory : Memory, inputValues : ArrayList, inputs : ArrayList<Typ
-constructIfStatements(ifStatement : String) : ArrayList<IfStatement>
-constructStates(curr : String, state : State) : void
-constructMemory(curr : String, memory : Memory) : void
-constructType(curr : String) : void
-constructModel(curr : String) : void

Figure D.4: The translator component of the ^{sp}XM2Visual tool, part 3.

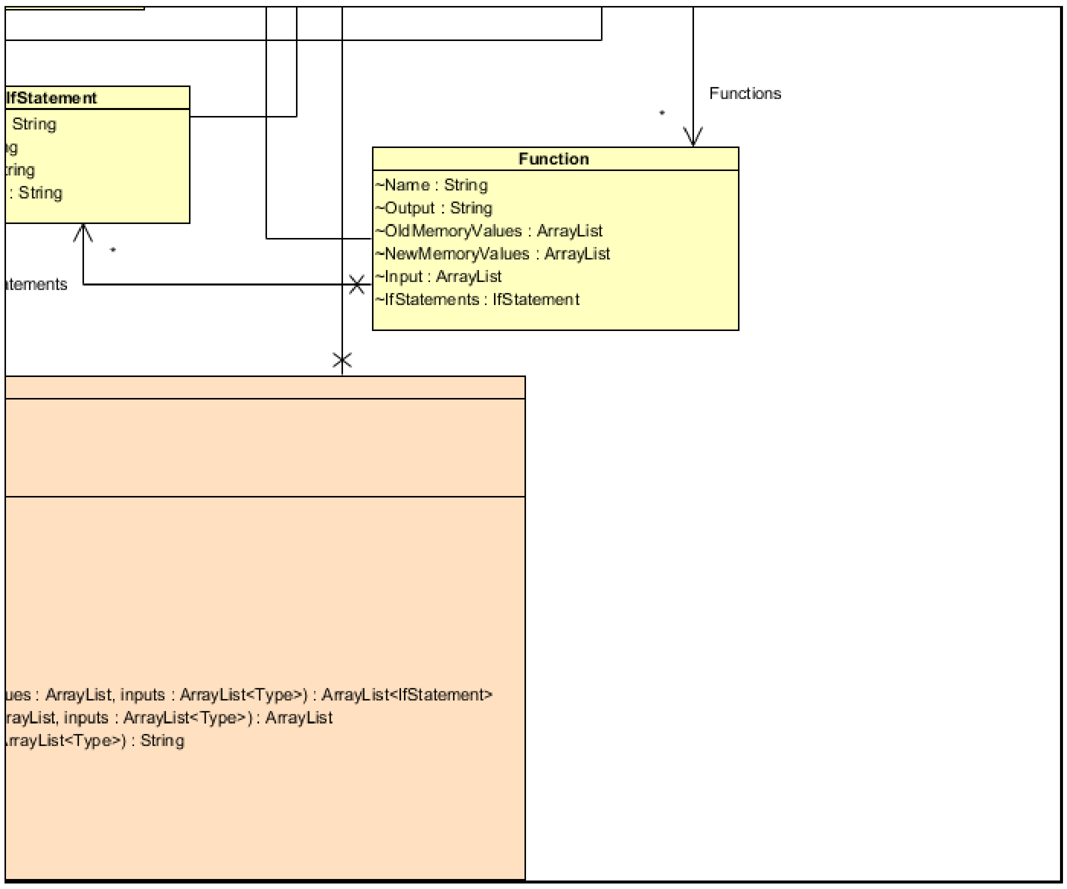
IfStatement
String
g
tring
: String
tements
*
Functions
*
Function
~Name : String
~Output : String
~OldMemoryValues : ArrayList
~NewMemoryValues : ArrayList
~Input : ArrayList
~IfStatements : IfStatement
ues : ArrayList, inputs : ArrayList<Type>) : ArrayList<IfStatement>
rayList, inputs : ArrayList<Type>) : ArrayList
rrayList<Type>) : String

Figure D.5: The translator component of the ^{sp}XM2Visual tool, part 4.

References

[1] J. R. Abrial. Modeling in Event-B. *System and software engineering*, 2010. 46

[2] C. Anderson. Linking micro to macro-level behavior in the Aggressor-Defender-Stalker game. *Proceedings of the Second International Workshop on the Mathematics and Algorithms of Social Insects*, pages 9–16, 2003. Atlanta, GA. 2, 95

[3] J. Beal, S. O. Dulman, K. Usbeck, M. Viroli, and N. Correll. *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, chapter Organizing the Aggregate: Languages for Spatial Computing. IGI Global, 2012. 13

[4] F. Bernardini and M. Gheorghe. Population P systems. In *Journal of Universal Computer Science*, pages 509–539, 2004. 23, 77

[5] B. W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1981. 42

[6] E. Bonabeau. Agent-based modeling: methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, pages 7280–7287, 2002. Washington, United-States. 46

[7] R. A. Brooks. Intelligence without representation. *Artificial Intelligence 47*, pages 139–159, 1991. 29

[8] L. Cardelli. Brane calculi. *Computational Methods in Systems Biology, International Conference (CMSB 2004)*, pages 257–278, May 2004. 14

[9] L. Cardelli and P. Gardner. Processes in space. In *CiE'10*, pages 78–87, Heidelberg, 2010. Springer-Verlag Berlin. 14, 71

[10] J. Casti. *Complexification.* Harper Collins, New York, 1994. 2

[11] Czech Technical Institute Agent Technology Center. Aglobe, 2011. URL `http://agents.felk.cvut.cz/aglobe/`. 14

[12] L. S. Chin, Science, and Technology Facilities Council (Great Britain). *FLAME-II: A Redesign of the Flexible Large-scale Agent-based Modelling Environment.* Technical report (Science and Technology Facilities Council (Great Britain)). STFC, 2012. URL `http://books.google.de/books?id=b3bRlwEACAAJ`. 38, 44, 59, 60

[13] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *Int. J. Software Tools Technol. Transfer*, 2:410–425, 2000. 45

[14] N. T. Collier and M. J. North. Repast SC++: A platform for large-scale agent-based modeling. Large-Scale Computing Techniques for Complex System Simulations, Wiley, 2011. (In Press). 53

[15] N. T. Collier, T. R. Howe, and M. J. North. Test-driven simulation development using Repast Simphony. *Proceedings of the North American Association for Computational Social and Organizational Science (NAACSOS)*, 2007 Annual Conference, June 2007. Emory University, Atlanta. 59

[16] P. Coveney and R. Highfield. *Frontiers of complexity.* Faber and Faber, London, 1996. 2

[17] C. Deissenberg, S. van der Hoog, and H. Dawid. EURACE: A massively parallel agent-based model of the european economy. *Applied Mathematics and Computation*, 204(2):541–552, October 2008. 60

[18] D. Dranidis, E. Ramollari, and D.Kourtesis. Run-time verification of behavioural conformance for conversational web services. *Seventh IEEE European Conference on Web Services*, pages 139–147, 2009. 12, 35

[19] D. Dranidis, K. Bratanis, and F. Ipate. JSXM: A tool for automated test generation. In *Software Engineering and Formal Methods*, volume 7504 of *Lecture Notes in Computer Science*, pages 352–366. Springer Berlin Heidelberg, 2012. 35, 36, 45

[20] G. Eleftherakis, P.Kefalas, and A.Sotiriadou. XmCTL: Extending temporal logic to facilitate formal verification of X-machines. pages 79–95, Analele Universitatii Bucharest, 2002. Matematica-Informatica. 44, 45, 96

[21] G. Eleftherakis, P. Kefalas, and A. Sotiriadou. Formal modelling and verification of reactive agents for intelligent control. In *Proceedings of the 12th Intelligent Systems Application to Power Systems Conference (ISAP03)*, Lemnos, Greece, September 2003. IEEE Power Engineering Society. 35

[22] M. D. Fraser, K. Kumar, and V. K. Vaishnavi. Strategies for incorporating formal specifications in software development. *CACM*, 37 (10):74–86, 1994. 1

[23] M. Gheorghe and F. Ipate. On testing P systems. In *Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 204–216. Springer Berlin Heidelberg, 2009. 46

[24] M. Gheorghe and G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000. 13, 14

[25] M. Gheorghe, F. Ipate, R. Lefticaru, and C. Dragomir. An integrated approach to P systems formal verification. *In Proc. CMC'10*, pages 225–238, 2010. 69

[26] M. Gheorghe, F. Ipate, and C. Dragomir. A kernel P system. *In Proc. BWMC10. Fénix Editora*, pages 153–170, 2012. 69

[27] M. Gheorghe, F. Ipate, R. Lefticaru, M. J. Pérez-Jiménez, A. Turcanu, L. Mierla, L. Valencia Cabrera, and F. M. Garcia-Quismondo. 3-Col problem modelling using simple kernel P systems. *International Journal of Computer Mathematics*, 90(4):816–830, 2013. 69

[28] J. Giavitto, C. Godin, O. Michel, and P. Prusinkiewicz. Computational models for integrative and developmental biology. *Technical Report 72-2002*, 2002. 14

[29] O. Gurcan, O. Dikenelli, and C. Bernon. Towards a generic testing framework for agent-based simulation models. *Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 635–642, Sept. 2011. Emory University, Atlanta. 59

[30] A. Helsinger, M. Thome, and T. Wright. Cougaar: a scalable, distributed multi-agent architecture. 2:1910–1917, 2004. 14

[31] M. Holcombe. X-machines as a basis for dynamic system specification. In *Software Engineering Journal*, pages 69–76, 1988. 1, 12, 30, 59

[32] M. Holcombe and F. Ipate. *Correct Systems: Building a Business Process Solution*. Springer, London, 1998. 1, 92

[33] G. J. Holzmann. The model checker SPIN. *IEEE IFans. on Software Engineering*, pages 279–295, 1997. 46, 68

[34] F. Ipate and M. Holcombe. Specification and testing using generalised machines: a presentation and a case study. pages 61–81. Software Testing, Verification and Reliability, 1998. 44, 45, 96

[35] F. Ipate and A. Turcanu. Modelling, verification and testing of P systems using Rodin and ProB. *In Proc. BWMC9. Fénix Editora*, pages 209–220, 2011. 69

[36] F. Ipate and A. Turcanu. Modeling, verification and testing of P systems using Rodin and ProB. *In Ninth Brainstorming Week on Membrane Computing*, pages 209–220, 2011. 46

[37] F. Ipate, M. Gheorghe, and R. Lefticaru. Test generation from p systems using model checking. *The Journal of Logic and Algebraic Programming*, 79 (6):350–362, 2010. 45, 46

[38] F. Ipate, R. Lefticaru, and C. Tudose. Formal verification of P systems using SPIN. *Int. Journal Found. Computer Science*, 22(1):133–142, 2011. 69

[39] F. Ipate, R. Lefticaru, and C. Tudose. Formal verification of P systems using SPIN. *Int. J. Found. Comput. Sci.*, 22(1):133–142, 2011. 45

[40] F. Ipate, R. Lefticaru, L. Mierla, L. Valencia Cabrera, H. Han, G. Zhang, C. Dragomir, M. J. Pérez-Jiménez, and M. Gheorghe. Kernel P systems: Applications and implementations. *In Proc. BIC-TA'13*, volume 202 of Advances in Intelligent Systems and Computing:1081–1089, 2013. 69

[41] C. B. Jones. *Systematic Software Development using VDM.* Englewood Cliffs. NJ: Prentice-Hall, 2 edition, 1990. 1, 12

[42] P. Kefalas and E. Kapeti. A design language and tool for X-machines specification. In D.I.Fotiadis and S.D.Nikolopoulos, editors, *Advances in Informatics*, pages 134–145, Singapore, 2000. World Scientific Publishing Company. 12, 15, 33, 34, 35

[43] P. Kefalas and I. Stamatopoulou. Towards modelling of reactive, goal-oriented and hybrid intelligent agents using P systems. In *Proceedings of the 11th International Conference on Membrane Computing*, pages 265–272, Berlin, Heidelberg, 2010. Springer-Verlag. 27, 29

[44] P. Kefalas, G. Eleftherakis, and E. Kehris. Modular modelling of largescale systems using communicating x-machines. *Proceedings of the 8th Panhellenic Conference in Informatics*, pages 20–29, 2001. 15, 34

[45] P. Kefalas, G. Eleftherakis, and A. Sotiriadou. Developing tools for formal methods. In *Proceedings of the 9th Panehellenic Conference in Informatics*, 2002. 1, 15, 44, 76

[46] P. Kefalas, G. Eleftherakis, and E. Kehris. Communicating X-machines: A practical approach for formal and modular specification of large systems. *Information and Software Technology*, 45:269–280, 2003. 12, 15, 34, 59, 96

[47] P. Kefalas, M. Holcombe, G. Eleftherakis, and M. Gheorge. A formal method for the development of agent based systems. In V.Plekhanova, editor, *Intelligent Agent Software Engineering*, pages 68–98. Idea Group Publishing Co., 2003. 1, 15

[48] P. Kefalas, I. Stamatopoulou, I. Sakellariou, and G. Eleftherakis. Transforming communicating X-machines into P systems. Journal of Natural Computing, Springer, 2009. 46

[49] S. Kripke. Semantical considerations on modal logic. 16:83–94, 1963. 46

[50] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. *In Proc. PAPM/PROBMIV'01 Tools Session*, pages 7–12, 2001. 68

[51] Telecom Italia Lab. Jade: Java Agent DEvelopment framework, 2011. URL `http://jade.tilab.com`. 14

[52] The Klavins Lab. Gro: The cell programming language. 2012. 14

[53] R. Lewin. *Complexity: Life on the edge*. Phoenix, London, 1995. 2

[54] S. Luke, G. C. Balan, L. Panait, C. Cioffi-Revilla, and S. Paus. MASON: A java multi-agent simulation library. *Proceedings of the Agent 2003 Conference*, 2003. 14, 44

[55] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Englewood Cliffs, NJ, 1993. 68

[56] D. E. Michael, G. G. William, K. Yoshio, and D. Notkin. Dynamically discovering pointer-based program invariants. Technical Report UW-CSE-99-11-02, University of Washington Department of Computer Science and Engineering, Seattle, WA, November 1999. Revised March 17, 2000. 68

[57] R. Milner. Communicating and mobile systems: The pi-Calculus. *CUP*, 1999. 14

[58] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm simulation system, a toolkit for building multi-agent simulations. Technical report, 1996. 14, 44

[59] S. Mirschel, K. Steinmetz, M. Rempel, M. Ginkel, and E. D. Gilles. Promot: Modular modeling for systems biology. *Bioinformatics*, 25(5):687–689, 2009. 14

[60] C. Myers, N. Barker, K. Jones, H. Kuwahara, C. Madsen, and N. Nguyen. iBioSim: a tool for the analysis and design of genetic circuits. 25, 2009. 14

[61] M. J. North, N. T. Collier, J. Ozik, E. Tatara, M. Altaweel, C. M. Macal, M. Bragen, and P. Sydelko. Complex adaptive systems modeling with Repast Simphony. Complex Adaptive Systems Modeling, Springer, Heidelberg, 2013. 14, 44, 53, 54, 60

[62] J. Odell, H. Parunak, and B. Bauer. Extending uml for agents. *Ann Arbor*, 1001:48-103, 1999. 14

[63] I. Petreska. Further material, 2011. URL `http://people.seerc.org/petreska/further_material.html`. 19

[64] I. Petreska. Tools for visual simulation of mas models. In *Proceedings of the 7th Annual SEERC Doctoral Student Conference (DSC 2012)*, pages 475–480, Thessaloniki, Greece, September 2012. 62, 102, 103

[65] I. Petreska and P. Kefalas. Towards novel approaches to modelling and verification of biologically inspired multi-agent systems. In *Proceedings of the 5th Annual SEERC Doctoral Student Conference (DSC 2010)*, pages 445–452, Thessaloniki, Greece, September 2010. 87, 101

[66] I. Petreska and P. Kefalas. Population P systems with moving active cells. In *Proceedings of the Twelfth International Conference on Membrane Computing (CMC12)*, pages 421–431, Fontainebleau, France, August 2011. 32, 52, 79, 85, 94, 102

[67] I. Petreska and I. Stamatopoulou. A comparative study of tools for visualisation of state-based spatial multi-agent models. In *Proceedings of the 2013 Balkan Conference in Informatics (BCI'13)*, pages 53–60, Thessaloniki, Greece, September 2013. 62, 103

[68] I. Petreska, P. Kefalas, and M. Gheorghe. A framework towards the verification of emergent properties in spatial multi-agent systems. In *Proceedings of the Workshop on Applications of Software Agents (WASA 2011)*, pages 37–44, Novi Sad, Serbia, July 2011. 32, 43, 52, 64, 68, 70, 95, 101

[69] I. Petreska, P. Kefalas, and M. Gheorghe. Informal verification by visualisation of state-based formal models. In *Proceedings of the 6th Annual SEERC Doctoral Student Conference (DSC 2011)*, pages 309–319, Thessaloniki, Greece, September 2011. 102

[70] I. Petreska, P. Kefalas, M. Gheorghe, and I. Stamatopoulou. Extending X-machines to support representation of spatial 2-d agents. In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence (ICAART 2012)*, pages 54–61, Vilamoura (Algarve), Portugal, February 2012. 85, 94, 102

[71] I. Petreska, P. Kefalas, M. Gheorghe, and I. Stamatopoulou. ^{sp}X-machines: Formal state-based modelling of spatial agents. In J. Filipe and A. Fred, editors, *Agents and Artificial Intelligence*, volume 358 of *Communications in Computer and Information Science*, pages 379–391. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36906-3. doi: 10.1007/978-3-642-36907-0_25. URL http://dx.doi.org/10.1007/978-3-642-36907-0_25. 70, 85, 94, 103

[72] M. Pogson, R. Smallwood, E. Qwarnstrom, and M. Holcombe. Formal agent-based modelling of intracellular chemical interactions. *Biosystems*, 85:37–45, 2006. 44

[73] M. Pogson, M. Holcombe, R. Smallwood, and E. Qwarnstrom. Introducing spatial information into predictive NF-kB modelling - an agent-based approach. *PLoS ONE*, 3(6):e2367, 06 2008. 71

[74] P. Prusinkiewicz and A. Lindenmayer. The algorithmic beauty of plants. 1990. 14

[75] G. Păun. *Membrane Computing: An Introduction*. Springer, Berlin, 2002. 13, 21, 22, 30

[76] G. Păun and G. Rozenberg. An introduction to and an overview of membrane computing. In *The Oxford Handbook of Membrane Computing*, pages 1–27, Oxford, 2010. Oxford University Press. 21, 22, 23

[77] G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford. Oxford University Press, 2010. 23

[78] S. Rahimi, M. Cobband D. Ali, M. Paprzycki, and F. Petry. A knowledge-based multi-agent system for geospatial data conflation. In *Journal of Geographic Information and Decision Analysis*, pages 67–81, 2002. 14

[79] W. Reisig. Petri nets: An introduction. In *EATCS Monographs on Theoretical Computer Science*, Berlin, 1985. Springer. 1, 12

[80] P. Richmond and D. Romano. Agent based GPU, a real-time 3D simulation and interactive visualisation framework for massive agent based modelling on the GPU. In *International Workshop on Supervisualisation 2008 (IWSV08)*, Kos Island, Greece, June 2008. 54

[81] D. A. Robertson. Agent-based modeling toolkits. *Academy of Management Learning and Education*, 4(4):525–527, 2005. 44

[82] F. J. Romero-Campero, J. Twycross, M. Camara, M. Bennett, M. Gheorghe, and Natalio Krasnogor. Modular assembly of cell systems biology models using P systems. In *International Journal of Foundations of Computer Science*, pages 427–442, 2009. 71

[83] R. Smallwood, M. Holcombe, and D. Walker. Development and validation of computational models of cellular interaction. *Journal of Molecular Histology*, 35:659–665, 2004. 38, 59

[84] L. P. Smith, F. T. Bergmann, D. Chandran, and H. M. Sauro. Antimony: a modular model definition language. 25(18), 2009. 14

[85] M. Spivey. *The Z Notation: A Reference Manual.* Englewood Cliffs. NJ: Prentice-Hall, 1989. 1, 12

[86] I. Stamatopoulou, M. Gheorghe, and P. Kefalas. *Modelling dynamic configuration of biology-inspired multi-agent systems with Communicating X-machines and Population P Systems*, volume 3365:389-401 of *LNCS*. Springer-Verlag, Berlin, 2005. 12, 27, 36

[87] I. Stamatopoulou, P. Kefalas, G. Eleftherakis, and M. Gheorghe. A modelling language and tool for Population P Systems. In *Proceedings of the 10th Panhellenic Conference in Informatics, PCI-05*, 2005. 22, 36, 37, 38, 81

[88] I. Stamatopoulou, P. Kefalas, and M. Gheorghe. OPERAS: A framework for the formal modelling of multi-agent systems and its application to swarm-based systems. In *ESAW*, pages 158–174, Berlin, Heidelberg, 2007. Springer-Verlag. 12, 13, 29, 30

[89] I. Stamatopoulou, I. Sakellariou, P. Kefalas, and G. Eleftherakis. OPERAS for social insects: Formal modelling and prototype simulation. *Romanian Journal of Information Science and Technology*, 11:267–280, 2008. 29, 30

[90] I. Trencansky and R. Cervenka. Agent modeling language (aml): A comprehensive approach to modeling mas. *Informatica Ljubljana*, 29(4):391, 2005. 14

[91] A. Turcanu and F. Ipate. Modelling, testing and verification of P systems with active membranes using Rodin and ProB. *In Proc. CMC'11*, pages 459–468, 2011. 69

[92] A. Turcanu, L. Mierla, F. Ipate, A. Stefanescu, H. Bai, and S. Coakley M. Holcombe. Modelling and analysis of E. coli respiratory chain. submitted. 69

[93] D. Walker, S. Wood, J. Southgate, M. Holcombe, and R. Smallwood. An integrated agent-mathematical model of the effect of intercellular signalling via the epidermal growth factor receptor on cell proliferation. *Journal of Theoretical Biology*, 9, 2003. 39

[94] U. Wilensky. NetLogo segregation model. *Center for Connected Learning and Computer-Based Modeling*, 1997. http://ccl.northwestern.edu/netlogo/models/ Segregation. 12, 53, 59, 60, 92

[95] U. Wilensky. NetLogo simulation software. *Center for Connected Learning and Computer-Based Modeling*, 1999. http://ccl.northwestern.edu/netlogo/. 12, 14, 44, 53, 59, 92

[96] M. Wooldridge. Agents and software engineering. *AI*IA Notizie XI(3)*, pages 31–37, September 1998. 10

[97] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995. 7, 8