AGENT-BASED METHODOLOGY FOR DEVELOPING AGROECOSYSTEMS' SIMULATIONS

ING. JORGE CORRAL AREÁN

TRABAJO DE TESIS
PARA LA OBTENCIÓN DEL TÍTULO
DE MAGÍSTER EN COMPUTACIÓN

CENTRO DE POSGRADOS Y
ACTUALIZACIÓN PROFESIONAL
INSTITUTO DE COMPUTACIÓN
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA

Enero 2011
Montevideo – Uruguay

Supervisor:
MSc. Ing. Daniel Calegari

Revisor:
Dr. Pierre Bommel

Tribunal:
Dr. Gustavo Betarte
MSc. Ing. Antonio Mauttone

Acknowledgements

I am very thankful (in no particular order)…

…To the EULACIAS Project Team and the Wageningen University and Research Centre (WUR) of The Netherlands for supporting me at the very beginning of this work, particularly Santiago Dogliotti and Walter Rossing. A special acknowledgement goes to Jeroen Groot for his always wise comments and revisions of the early chapters of this thesis. I'm in debt with all of you.

…To the Instituto Plan Agropecuario (IPA) for letting me be part not only of this FPTA project but of the team itself. I don't want to forget any of you here: Danilo Bartaburu, Esteban Montes, Emilio Duarte, Marcelo Pereira, Francisco Dieguez and Paulina Altieri, and also to its Board of Directors. A special acknowledgement goes to Hermes Morales Grosskopf because he introduced me not only to MAS but to agronomy and showed me there are many people that no longer identify themselves with just one discipline.

…To the French Centre de Coopération Internationale en Recherche Agronomique pour le Développement (CIRAD) for letting me be part of several research projects and getting more involved in the MAS topics. A special acknowledgement goes to Jean Pierre Müller for our fruitful discussions here in Uruguay, to Jean Francois Tourrand for constantly inviting me to projects and meetings (hoping you will continue to do so!) and of course, to Pierre Bommel, who taught me the vast majority of what I know about MAS, for his constant disposition to help and support, giving me the necessary confidence and trust to follow his steps in every project in which we participate together, including the revision of this thesis.

…To my supervisor, Daniel Calegari, for the delicate balance that he kept between giving me the freedom I needed and stating the results and milestones that he expected, and also for his accurate corrections and invaluable guidance. Also to the Faculty of Engineering (FING, UdelaR, including my colleagues at the COAL group) for supporting me in this adventure of combining computer science and agronomy.

…To my family and friends, since there is no big project that one can embrace without their emotional and spiritual support.

I would also like to acknowledge that this work was partially funded by the Instituto Nacional de Investigación Agropecuaria (INIA) through the FPTA-funded project that is part of Chapter 6 of this thesis.

I know I am probably missing someone, so forgive me in advance.

I am very, extremely, grateful to all of you for your *trust* and *support* without which this work would have never been possible.

And last, but not least, this thesis is dedicated to my daughter, María Paz: I love you from the bottom of my heart.

Abstract

The agent-based modeling (ABM) approach allows modeling complex systems, involving different kinds of interacting autonomous agents with heterogeneous behavior. Agro-ecosystems (ecological systems subject to human interaction) are a kind of complex system whose analysis and simulation is of interest to several disciplines (e.g. agronomy, ecology or sociology). In this context, the ABM approach appears as a suitable tool for modeling agro-ecosystems, along with a corresponding agent-oriented software engineering (AOSE) methodology for the construction of the simulation. Nevertheless, existing AOSE methodologies are general-purpose, they have not yet accomplished widespread use, and clear examples of applications to agro-ecosystems are hard to find. This thesis sets the ground for a new software development methodology for developing agro-ecosystem simulations based on the ABM approach as well as on these already existing AOSE methodologies, but tailored to tackle specific agro-ecosystem features.

Keywords: Multi-Agent Systems, Agro-Ecosystem, Software Development, Simulation.

Resumen

El enfoque de modelado basado en agentes (ABM) permite el modelado de sistemas complejos en los que interactúan diferentes tipos de agentes autónomos con comportamientos heterogéneos. Los agro-ecosistemas (sistemas ecológicos sujetos a la presencia humana) son un tipo de sistema complejo cuyo análisis y simulación resulta de interés para diversas disciplinas (ej.: agronomía, ecología o sociología). En este contexto, el enfoque ABM aparece como una herramienta adecuada para el modelado de agro-ecosistemas, junto con una correspondiente metodología de desarrollo de software también orientada a agentes (AOSE) para la construcción de dicha simulación. Si bien ya existen metodologías AOSE, éstas son de propósito general, no han logrado un amplio uso y ejemplos claros de aplicaciones a agro-ecosistemas son difíciles de encontrar. Esta tesis establece los fundamentos para crear una nueva metodología de desarrollo de software basada en el enfoque de agentes para el desarrollo de simulaciones de agro-ecosistemas, basándose en las metodologías AOSE ya existentes, pero personalizada para soportar las características específicas de los agro-ecosistemas.

Palabras clave: Sistemas Multi-Agente, Agro-ecosistema, Desarrollo de Software, Simulación.

Table of Contents

1. Introduction

Many of the current world challenges and opportunities (e.g. globalization, sustainability, terrorism, epidemics or climate change) can be seen as complex systems [Miller and Page, 2007]. Understanding the components, behavior and interactions in these systems is the first step to whatever analysis is needed about them.

Agro-ecosystems are natural ecosystems subject to human interaction. Therefore they are composed of a natural sub-system and a social (or human) sub-system. Even though not at global scale, agro-ecosystems can be seen as a case of complex systems that can represent altogether the natural and human aspects, as well as their interactions and feedback over time. However, the possibility for direct experimentation of such systems is rare, if not impossible, so the need for modeling and simulation becomes relevant.

Several approaches or methodologies can be used for modeling an agro-ecosystem. In particular, the agent-based modeling (ABM) approach appears as a suitable tool for this purpose since they can be used for modeling complex systems. On the other hand, simulating generally means developing a software system, in this case, representing the agent-based model. This requires the use of some agent-oriented software engineering (AOSE) methodology. Several AOSE methodologies are currently available for guiding a programmer in developing software following the ABM approach. However, they are general-purpose methodologies, so the programmer is faced with a trade-off between using an already existing one and not leveraging the specificities of agro-ecosystems, or to follow an ad-hoc methodology that pays detail to those features. Up to our knowledge there are no specific AOSE methodologies for this purpose. Nevertheless, there are some related work [Le Page and Bommel, 2005] that address the simulation of agro-ecosystems using an ABM approach but without a methodological framework behind it.

The overall objective of this thesis is to contribute to the development of an agent-based methodology for developing agro-ecosystem simulation software.

The work presented here aims at contributing to the development of such a methodology, but it does not attempt to develop a fully comprehensive one. Many topics considered relevant for a software development methodology, such as resource, time and risk management, budget and effort estimation, team work, and testing, among others, are not included in this thesis. Only those aspects directly related to constructing a program are considered, such as the analysis, design and implementation phases, along with their artifacts and techniques. This is why this thesis uses the term *methodological framework* in order to distinguish it from a fully-featured methodology.

The focus on the agent-based approach comes from its close relation to agro-ecosystems, more specifically because the latter can naturally be modeled using the former.

The interest in agro-ecosystems comes from my involvement in several agronomical research projects in the years before starting this thesis. Even though I played the role of "pure" software engineer, the multidisciplinary teams in which I had the honor to work, the new frontiers that these people showed me, the new real-world and real-people problems I was faced to, and the chance to re-focus my previous expertise into these topics, all of this showed me a window of opportunity ready to be opened.

Because of the very nature of the systems under study (natural and human), there is a need for simulating different scenarios and test various hypothesis, and in order to do this, we must count not only on software tools and packages to assist us, but also on methodological tools to guide the process.

Therefore, this thesis' specific objectives are the following:

- Present the state-of-the-art in the multi-agent systems (MAS) field that, apart from relating the necessary concepts, shows a clear separation of the different areas of study that are involved (i.e. complex systems, agents, multi-agent systems, artificial intelligence), including a solid line of thought that clearly justifies the use of multi-agent systems and the agent-based modeling (ABM) approach to modeling (and later simulating) an agro-ecosystem (Chapter 2).
- Present the state-of-the-art in the agent-oriented software engineering (AOSE) methodologies and show that none of them is specific to the modeling and simulation of agro-ecosystems (Chapter 3).
- Propose a methodological framework that allows developing agro-ecosystem software simulations using the agent-based modeling approach, leveraging existing AOSE methodologies where possible, as well as leveraging software developers' object-oriented programming (OOP) and Unified Modeling Language (UML) knowledge (Chapter 4).
- Verify that all relevant agro-ecosystem's features are covered by the proposed methodological framework (Chapter 5).
- Achieve an initial validation of the methodological framework by applying it to a real-world complex case study (Chapter 6).

Having all these objectives accomplished, this thesis' contributions are the following:

- The only methodological framework for developing agro-ecosystem simulations using the agent-based modeling approach. Up to our knowledge, there is no other methodology (nor methodological framework) that copes with this problem.
- The only case in which the ABM approach was applied to a Uruguayan agro-ecosystem problem, in this case the draught phenomena in the basalt region and a study on how it affects cattle breeders and their draught strategies.
- The possibility to get regular OOP+UML software developers to quickly start writing agro-ecosystem simulation software, without previous knowledge of MAS. This becomes increasingly important because it reduces the time and skills required to get into a multidisciplinary team, especially when software developers are hard to find.
- The possibility to formalize a methodology that allows developers (as stated above) as well as experts (e.g. agronomists) and producers to jointly participate in the development of a simulation, achieving a truly interdisciplinary work.
- A clear separation of the areas of study and fields that are involved. Usually the papers around the topic of ABM and agro-ecosystems do not clearly separate the fundamental concepts involved, leading to misunderstandings, which increase the time and effort to getting involved in the topic.

The structure of this thesis is summarized in **Figure 1**:

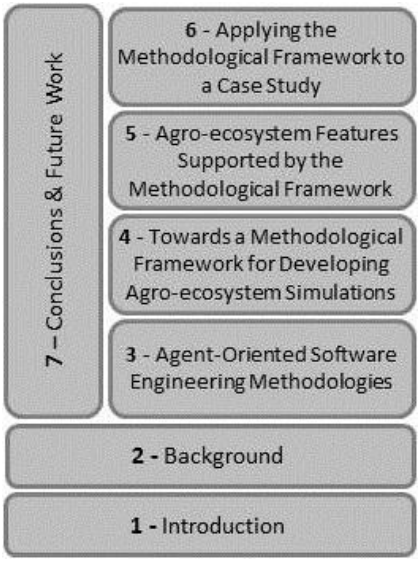


Figure 1: Chapter overview of this thesis.

Chapter 2 presents the main concepts and state-of-the-art regarding complex systems, agents, multi-agent systems, agro-ecosystems and how they relate to each other. It also serves as the necessary background knowledge needed to understand the rest of the thesis.

Chapter 3 presents the main ideas behind Agent-Oriented Software Engineering (AOSE) and a summary and comparison of the most well-known AOSE development methodologies.

Chapter 4 presents the methodological framework that is proposed in this thesis for developing agro-ecosystem simulation software based on the agent-based modeling approach and on ideas from the existing AOSE of Chapter 3.

Chapter 5 presents those features that characterize an agro-ecosystem and shows how they are covered by the methodological framework proposed in Chapter 4.

Chapter 6 applies the methodological framework to a case study.

Chapter 7 concludes this thesis and presents possible future work.

2. Background

This chapter introduces the main concepts that will be elaborated in this thesis and presents a state-of-the-art regarding complex systems, agents, multi-agent systems, agro-ecosystems and how they relate to each other. It assumes no previous knowledge of concepts related to complex systems, agent-based modeling or agro-ecosystems. This chapter is an excerpt of **Appendix A**.

Section 2.1 presents fundamental concepts and principles related to complex systems which are the general context for agro-ecosystems presented in Section 2.2. Section 2.3 introduces the agent-based modeling approach, and Section 2.4 shows the applicability of the agent-based modeling approach to complex systems in general and to agro-ecosystems in particular. Section 2.5 compares the Object-Oriented (OOP) and Agent-Oriented (AOP) programming paradigms. Finally, the chapter ends with some concluding remarks in Section 2.6.

2.1 Complex Systems

In natural sciences and computer science, the most common approach to understand or analyze a system is based on a decomposition of the system into its elementary parts and the isolated and in-depth study of these in order to understand the whole, i.e. the reductionism of Descartes.

The systemic approach starts by first examining and understanding the relations between the different elements of the system. The systemic approach is a general theory since its principles can be applied to any discipline or area. One of the most common ways for addressing the systemic approach is to say that 'the whole is more than just the sum of its parts' or in the words of [Miller and Page, 2007]: "*The field of complex systems challenges the notion that by perfectly understanding the behavior of each component part of a system we will then understand the system as a whole*".

Both complicated and complex systems are composed of a large number of interacting elements, but two properties set a complex system apart from one that is merely complicated: emergence and self-organization. Emergence is the appearance of behavior that could not be anticipated from the knowledge of the parts of the system alone [CSIRO, 2008]. Moreover, the newly emerged properties can in turn feedback to the original lower-level entities, entering a feedback loop where each element (micro/marco level) interacts. Self-organization means that there is no external controller or planner engineering the appearance of the emergent features; they appear spontaneously [CSIRO, 2008]. The motivation for studying complex systems is that many of current opportunities and challenges (globalization, sustainability, terrorism, epidemics, climate change) are complex. Each of these domains consists of a set of diverse entities and actors that dynamically interact, and are immersed in a sea of feedback [Miller and Page, 2007]. **Figure 2** shows a schematic representation of a complex system based on [Parrott, 2002].

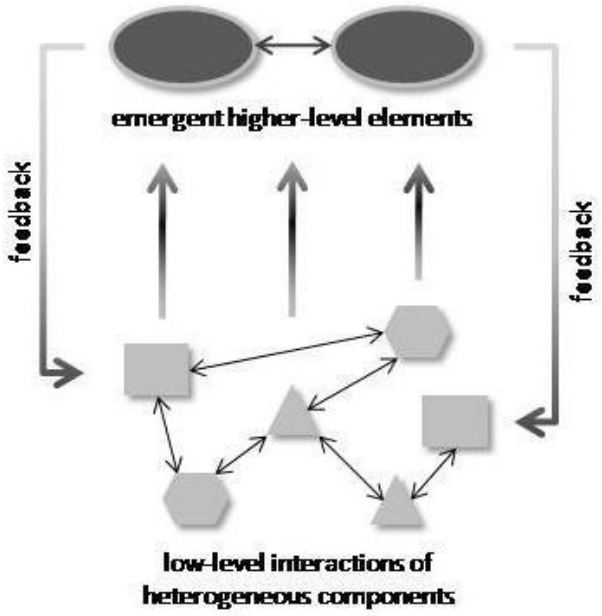


Figure 2: Schematic Conceptual Model of a Complex System.

Complex Adaptive Systems (CAS) are defined as systems that are capable to adapt and self-organize in response to perturbations or distortions in the environment or by the result of certain interrelations between the elements. System adaptation is ultimately concerned with the adaptation of each individual element of the system, since there is no centralized control and therefore no single 'object' that represents the entire system.

2.2 Agro-Ecosystems

This section introduces the notion of agro-ecosystem and related concepts. Also, agro-ecosystems are presented as a special case of complex systems. Finally, the importance of modeling and simulation of agro-ecosystems is highlighted.

2.2.1 What is an Agro-ecosystem?

An agro-ecosystem is the human manipulation and alteration of ecosystems for the purpose of establishing agricultural production [Gliessman, 1997]. Agro-ecosystems result from the interplay between endogenous biological and environmental features of the agricultural fields and exogenous social and economic factors, and are delimited by arbitrarily chosen boundaries. They are semi-domesticated ecosystems that fall on a gradient between ecosystems that have experienced minimal human impact, and those under maximum human control [Hecht, 1987].

According to [Odum, 1984] the four major characteristics of agro-ecosystems are:

- They include **external sources** of energy like human, animal or fuel energy to enhance productivity of particular crops;
- **Diversity** may be reduced compared with natural ecosystems (those with no human intervention);
- The dominant animals and plants are under **artificial selection** rather natural selection; and
- The **system controls** are external rather than internal via subsystem feedback, in the sense that the natural resources are no less dependent on natural factors because of human intervention.

Even though this is a good starting point according to [Hecht, 1997] it does not reflect agro-ecosystems that can be relatively diverse and its lack of attention to the social determinants of agriculture limits its explanatory power. Agricultural systems are human artifacts, and the determinants of agriculture do not stop at the boundaries of the field. Agricultural strategies respond not only to environmental, biotic[1], and cultivar constraints, but also reflect human subsistence strategies and economic conditions [Hecht, 1987; Ellen, 1982]. This stresses the importance of social factors like labor availability, access and conditions of credit, subsidies, perceived risk, price information, association obligations, family size, and access to other forms of livelihood are often critical to understanding the logic of a farming system [Hecht, 1987].

An agro-ecosystem thus has physical parts with particular relationships (the *structure* of the system) that together take part in dynamic processes (the *function* of the system) [Gliessman, 1997]. The structure can be viewed as organized in several levels, ranging from individual elements such as organisms or crops up to regions, landscapes or entire countries.

Any system uses its processes and resources to convert its inputs into its outputs. Concerning the resources commonly found in agro-ecosystems, [Norman, 1979] suggests the following classification:

- **Natural resources**: the given elements of land, water, climate and natural vegetation that are exploited by the farmer.
- **Human resources**: the people who live and work within the farm and use its resources for agricultural production, based on their traditional or economic incentives.
- **Capital resources**: the goods and services created, purchased, or borrowed by the people associated with the farm to facilitate their exploitation of natural resources for agricultural production.
- **Production resources**: the agricultural output of the farm such as crops and livestock. These become capital resources when sold, and residues (crops, manure) are nutrient inputs reinvested in the system.

[1] Associated with or derived from living organisms (English On-Line Dictionary).

Several of all the previous concepts are summarized in **Figure 3**, based on [Altieri. 1995] and [Briggs and Courtney, 1985]:

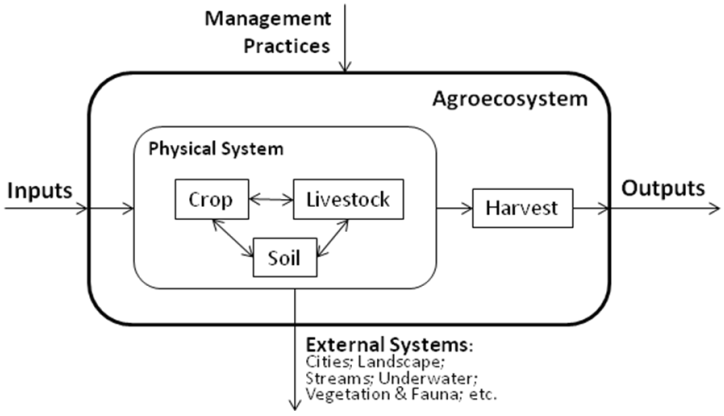


Figure 3: Representation of an Agro-ecosystem.

It is worth noting that human intervention is represented by the inputs but mainly by the management practices executed by farmers.

Other concepts that are also commonly associated to agro-ecosystems include natural processes (like erosion), the landscape to which the agro-ecosystem belongs, market prices and prices evolution (regarding for example international crop prices), and government policies (which may affect virtually any part of the agro-ecosystem).

2.2.2 Agro-ecosystems as Complex Adaptive Systems

Any ecosystem by itself can already be considered as a case of complex adaptive system, considering its various components, organization levels, micro and macro interactions and their feedbacks, and even more if including a social subsystem (with heterogeneous behaviors) as in the case of agro-ecosystems.

The interactions between the social and natural subsystems inside an agro-ecosystem, such as farmers' practices affecting natural resources, are examples of local interactions at a small scale which may produce negative or positive feedbacks affecting in turn the decision-making of social actors. Repeating these interactions on a daily basis can produce emergent properties and new organizations across the agro-ecosystem in the long-term, like a change in soil quality as a long-term consequence of the adoption of certain management practice or the emergence of a certain land-use pattern.

Aside from natural processes, also the interactions between the social actors can lead to complex behaviors like, for example, communications between neighboring producers which can affect each producer's performance (e.g. knowing about certain new market because of participation in an organization can lead to improvements in the income of those producers that receive the correct information at the right moment).

2.2.3 The Importance of Simulation in Agro-ecosystems

Since these coupled human-natural systems cannot be manipulated and tested as other systems, due to scale and resource difficulties (setting up farms managed by people over decades just to see the effects) the possibility of simulating them is crucial.

The objective of simulating these kind of systems is not for trying to figure out exactly what will happen in the future if the present conditions are X, the behavior of each part is Y and the evolution of certain parameters is Z. In contrast, simulation in these contexts should have the objective of prospection of scenarios were the interest is not in the 'fortune-teller' features of the simulation but on discovering possible outcomes under certain conditions and being able to easily modify these conditions and check again the outcomes (as in a virtual laboratory). Another objective can be to deepen the understanding (and also possibly learning/teaching) of a coupled human-natural system, since the more micro/macro level study of each part as well as their interactions require an in-depth understanding of them in order to obtain a simulation that is close to reality. Yet other objective can be to explore the consequences of manipulating certain system in order to understand how the different parts will co-exist in the future due to that manipulation (e.g. how does a water shortage affects the productivity of a farm, or how does certain economic policy impacts in the long term).

The next section presents an approach that can be used to develop such simulations in the form of software systems that will be able to simulate the different parts of an agro-ecosystem and monitor (observe) certain parameters over time in order to analyze their evolution.

2.3 Agent-Based Modeling

Agent based modeling (ABM) is a computer science approach that enables the simulation of heterogeneous populations of interacting individuals or agents, in many cases in a non-agent **environment**, which can also contain non-agent passive **objects** (commonly known as resources). The agents can exhibit a set of different behaviors, and the selected behavior is dependent on the local interactions with other individuals in their **neighborhood** and the state of the environment, thus the agents may be adaptive. The mechanism of selection of behavior can range from simple procedural logic to highly sophisticated **reasoning**. The repertoire of behaviors can be fixed or extensible, and the latter implies that the agents should be able of **learning**. The intelligence of the agent depends on its abilities to reason and to learn. The adaptive behavioral patterns enable self-organization of the population and can result in emergent phenomena. Consequently, the ABM approach is suitable to address complex adaptive systems. These concepts will be further studied in the following sections.

2.3.1 Definition of Agent

There is no consensus on the definition of agents, but the most used definitions were proposed by Wooldridge and Jennings (1995) and Ferber (1999).

A definition that supports the computer science focus presented in this thesis is proposed by [Wooldridge, 2008]: "*An agent is a computer system that is capable of independent action on behalf of its user or owner, figuring out what needs to be done to satisfy design objectives, rather than constantly being told [what to do]*". In this definition, the word 'independent' refers to agent autonomy, capable of acting independently and exhibiting control over its internal state. Thus an agent is a computer system capable of autonomous action in some environment in order to meet its design objectives [Wooldrige, 2002&2008] (Section 2.3.3 further discusses agents' environments).

According to [Wooldridge, 2002] an intelligent agent is a computer system capable of flexible autonomous action in some environment[2] and proposes the following properties in order to let an agent show intelligent behavior: reactive, proactive and social.

- **Reactive**: A reactive system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful) [Wooldridge, 2002]. Even though a changing environment makes agent design harder, it allows the representation of much more complex (and interesting) situations.
- **Proactive**: Since agents should be able to *do* something according to their design objectives (goals) they should have some way to direct their behavior towards those goals. Proactiveness means generating and attempting to achieve goals, not driven solely by events as in the case of reactivity but by actually taking the initiative [Wooldridge, 2002].
- **Social**: The real world is a multi-agent environment. As [Wooldrigde, 2002] states: "we cannot go around attempting to achieve goals without taking others into account". Social ability in agents is the ability to interact with other agents via some kind of agent-communication protocol or language, and perhaps also cooperate or negotiate with others.

In order to formalize some of the previous topics and to simplify future discussions without losing generality, the following concepts are introduced, which were extracted from [Wooldridge, 2002] and [Lind, 2008]. In order to let agents react (or take the initiative) according to changes in the environment, agents must perceive their environment and have some way to act upon it, after reasoning what to do. This leads agent to a Perceive/Reason/Act cycle, shown in **Figure 4**.

[2] One of the best known examples of agents are robots. From industry manufacturing to NASA explorers and soccer-playing, robots can be seen as a "*physical instantiation of an agent*" according to Wooldridge. Being their objective to ensemble a car, collect and analyze rocks or score a goal, being alone or in groups, each of these share, to more or less extent, the properties discussed here.

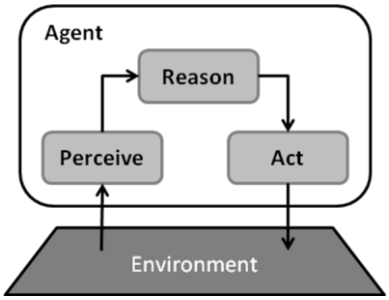


Figure 4: General representation of an agent with its environment.

2.3.2 Agent Architectures

Since there are multiple ways to achieve this Perceive/Reason/Act cycle, there are accordingly different agent architectures that represent and implement this concept in different ways and to different degrees. The term 'agent architecture' refers to the software architecture for the decision-making agents in the environment, as well as their internal mechanisms and representations [Wooldridge, 2002; Bousquet & Le Page, 2004].

Two issues should be taken into consideration regarding this term: first that in the context of agents, architectures are generally referred as being abstract (closer to reality) or concrete (closer to implementation); and second that several authors differ on the classification of agent's architectures.

The following classification of abstract architectures[3] is given by [Wooldridge, 2002]:

- **Deliberative Architectures** (also known as Symbolic or Logical AI) that manages explicit representations of desires (goals), beliefs (what the AI agent knows), intentions (what it wants to do), actions (what it does) and uses abstract reasoning tools. Deductive and deliberative agents appear in these architectures as well as the BDI Architecture (Belief, Desire, Intention)[4];
- **Reactive Architectures** that enable intelligence without having explicit representations, or abstract reasoning, but as an emergent property of certain complex systems. In this kind of architectures the agent has no previous knowledge and simply reacts based on a set of rules; and
- **Hybrid Architectures** that include a deliberative as well as a reactive component.

2.3.3 Definition of Multi-Agent System

After introducing what an agent is and considering agents that operate alone, the agency theory can be extended to let agents interact within a so-called Multi-Agent System (MAS). According to [Wooldridge, 2002]: "*A Multi-Agent System consists of a number of agents which interact with one-another. […] To successfully interact, they will require the ability to cooperate, coordinate, and negotiate with each other.*"

In MAS there is no central control and all information and control is distributed among the various agents. **Figure 5** presents an illustration of a MAS (based on [Wooldridge, 2002] and [Ferber, 1999]).

[3] In order to be able to implement software using such architectures, more detail and design must be considered.

[4] BDI architectures resemble human decision-making: belief is what the agent knows (its experience), desire is what the agent is willing to do, and intentions are desires plus the commitment to achieve them. In these architectures agents spent time not only doing things but also deliberating about what they should do. After choosing a plan to execute according to their current intentions, they are able to constantly introspect about the validity of such plan at each step of it, and if necessary, change the plan in order to better accomplish their goals. However, [Deffuant et al., 2003] explain that BDI models are not necessarily more realistic than the simple models and BDI architectures do not necessarily rely on robust scientific basis and do not derive from neuroscience precepts, nor psychology and neither of philosophy.

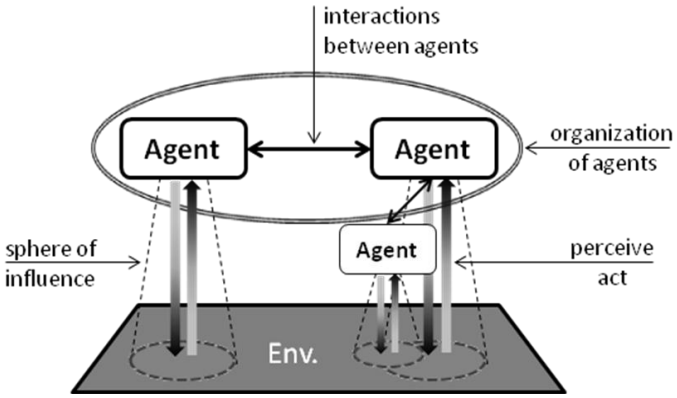


Figure 5: A Multi-Agent System.

As previously discussed, agents are able to perceive part (or all) of the environment and are also able to act upon part (or all) of it depending on the type of environment (refer to Appendix A for MAS environments). This can be represented by spheres of influence that determine the portion of the environment that one agent is able to interact with. Interactions between agents allow agents to cooperate, coordinate and negotiate as needed. Finally, interacting agents can also be organized to form higher-level organizational units.

To these concepts [Ferber, 1995] adds the concept of object as being a passive entity subject to manipulation (consumption and production) by agents (which in turn are defined as a subset of those objects but with the ability to be active). All of these are also defined to be located, meaning that at each time it is possible to locate each object and agent within the environment. Finally Ferber introduces the idea that each agent possess its own (internal) representation of the environment, of the objects and of other agents.

In this thesis the term Agent-Based Modeling will be used indistinctively from the term Multi-Agent System since the first is no more than the process of modeling a certain reality using agents in order to build a multi-agent system[5].

A concept that enables further elaboration on the agency theory in a multi-agent environment is the one of a *role*. A role is the functional or social part which an agent, embedded in a multi-agent environment, plays in a process like problem solving, planning or learning [Lind, 2008]. Roles are a useful abstraction considering the widely accepted meaning of the term in the real world that can be used to help describe and understand the system by describing the consistency of an agent's behavior within that system (as well as within the organization to which the agent belongs).

In MAS agents interact with each other, and this interaction can lead to cooperation (when different agents share objectives) or negotiation (when the interests are not aligned). Negotiation is the process of reaching agreements on matters of common interest [Wooldrigde, 2002].

2.4 Agent-Based Modeling of Complex Systems

This section presents the applicability of the agent-based modeling approach to complex adaptive systems, and then discusses the suitability of the approach to agro-ecosystems as special case of complex systems.

2.4.1 Applicability of Agent-Based Models to Complex Adaptive Systems

Besides from the similarities between complex adaptive systems and the agent-based modeling approach that could already be noticed, this section further justifies the use of the latter to represent the former (refer to Appendix A for more reasons).

The following four points help clarifying why the ABM approach is suitable for representing CAS:

- **Regarding Emergence**: ABMs allow to define the low-level behavior of each individual agent in order to let them interact (over time and space) to see whether some emergent property arises or not, and if it does, under which circumstances.

[5] This is also supported by [Bousquet and Le Page, 2004]: "*Recently, several researchers have started to use multi-agent systems, also called agent-based modeling, in different fields.*"

- **Regarding Self-Organization**: ABMs do not have any kind of central intelligence that governs all agents. On the contrary, the sole interaction among agents along with their feedbacks is what ultimately 'controls' the system. This lack of a centralized control is what enables (and enforces) its self-organization.
- **Coupled Human-Natural Systems**: ABMs allow considering together both, social organizations with their human decision-making with biophysical processes and natural resources. This conjunction of subsystems enables ABMs to explore the interrelations between them, allowing analyzing the consequences of one over the other.
- **Spatially Explicit**: the feature of ABMs of being able to spatially represent an agent or a resource is of particular interest when communications and interactions among neighbors is a key issue. This can either imply some kind of internal representation of space or even the use of a Geographical Information System (GIS) with real data. This feature is of special interest in the case of agro-ecosystems.

The following Section presents how some special features of agro-ecosystems can be modeled using an ABM approach.

2.4.2 Modeling of Agro-Ecosystems using an ABM Approach

Section 2.2.2 presented agro-ecosystems as a case of complex adaptive systems, while Section 2.4.1 showed the suitability of the ABM approach to model complex adaptive systems. Nevertheless, this section provides some views on the topic by different authors working on ABM approach to agro-ecosystems and presents some specific features of agro-ecosystems that can also be tackled with it.

According to [Ferber, 1999, p. 36]:

> *"Multi-agent systems bring a radically new solution to the very concept of modeling and simulation in environmental sciences, by offering the possibility of directly representing individuals, their behavior and their interactions […] it is thus possible to represent a phenomenon as the fruit of the interactions of an assembly of agents with their own operational autonomy."*

The relations and heterogeneity in the social part of the MAS depend on the agent's social **neighborhood**, whereas the variability from the physical and ecological point of view can be achieved by considering spatial heterogeneity of the **environment** in the MAS.

The addition of spatially explicit features is especially important in agro-ecosystems where spatial heterogeneity can be relevant, for example by means of different land-uses, soil qualities, natural resources, etc. on each land-unit. Just as land-units can be organized to form higher-level spatial elements, social networks can be formed between agents or can emerge as a consequence of agent's behavior and interactions. What makes ABMs rather unique is that both of these organizational dimensions can be considered together, allowing exploring the consequences of one over the other.

Finally, it is worth noting that even though agents, from the ABM point of view, can be any autonomous and goal-driven entity, from the agro-ecosystems perspective agents are generally aspects of human societies or animal populations that organize among themselves, interact with the environment, and they are affected in their decisions by that environment [Matthews, 2006]. **Figure 6** shows these relationships similarly as **Figure 2** (in Section 2.1) showed an abstract complex system:

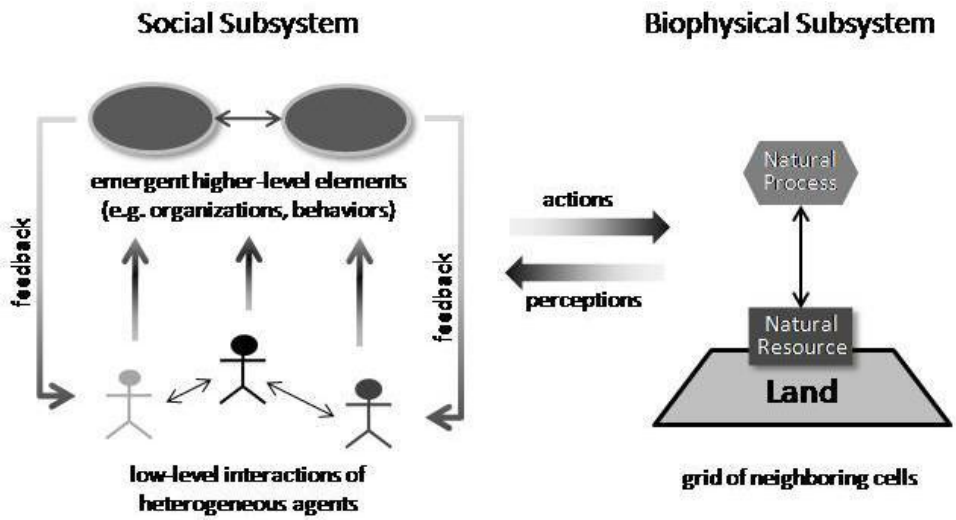


Figure 6: Model of an Agro-Ecosystem using an ABM Approach.

The actual implementation of an agent-based model is ultimately about developing a computational model, that is, software. Therefore, in order to analyze, design, implement and every other step involved in developing such systems, certain software development methodology must be used. The following section compares the agent and object-oriented paradigms. Appendix B completes this comparison by comparing the AOSE and OOSE methodologies.

2.5 Comparison Between Agent-Oriented and Object-Oriented Paradigms

In 1989 Yoav Shoham coined the term *agent-oriented programming* [Shoham, 2010] and defined it as a new programming paradigm based on a societal view of computation. This societal view implies (according to Shoham) ascribing mental qualities to agents such as beliefs, capabilities, choices and commitments (in the same way than BDI architectures propose to conceive agents in terms of beliefs, desires and intentions).

The idea behind these mental qualities includes the use of mental constructs to design computational systems. This is the main difference between AOP and OOP: while OOP uses abstractions based on objects which are generic *entities* that comprise *identity* (a property inherent to all objects that allows to differentiate them), *state* (attributes by which the object can be characterized) and *behavior* (operations that the object can be asked to do by means of messages), AOP considers abstractions based on *human societies* and *mental properties*.

Another important difference between both approaches is the autonomy that agents have. While in OOP objects just reply to messages sent from other objects in a predefined way, in AOP agents are supposed to be proactive and initiate actions over themselves or over other agents.

The following table summarizes some of the differences between OOP and AOP:

	OOP	**AOP**
Basic unit	Object (instance of a Class)	Agent instance of an Agent Type
Constraints defining the state of the basic unit	None	Mental properties (beliefs, desires, etc.)
Autonomy	Not necessarily present	May be present
Types of messages (interaction)	Unconstrained	Speech acts (inform, request, etc.)
Concept of role	Not necessarily present (comparable to using interfaces)	May be present

Table 1: Comparison between OOP and AOP paradigms.

This short list of differences illustrates that even though OOP and AOP have important similarities, these arise because AOP can be seen as a *specialization* of OOP, a new paradigm thought as constraining the extremely general-purpose of the OOP paradigm and aligning those constraints to mental and societal properties.

The value (and innovation) of AOP is in providing useful abstractions for understanding and developing systems in terms of agents and societies of agents. In words of [Jennings, 2001]: "*When designing software, the most powerful abstractions are those that minimize the semantic gap between the units of analysis that are intuitively used to conceptualize the problem, and the constructs present in the solution paradigm.*

For a comparison between OOSE and AOSE methodologies, refer to Appendix B.

2.6 Final Remarks

This chapter introduced the fundamental topics needed as background for the rest of this thesis. For a deeper state-of-the-art, refer to Appendix A. This chapter also showed that the systems approach along with concepts such as complexity and emergence provide a suitable way to conceptualize agro-ecosystems and their related problems at various scales, and that the agent-based modeling approach provides a way to operationalize these concepts into computational models that allow for simulation when direct experimentation is not possible. However, it is imperative for the software engineering discipline to provide the necessary guidance for developing such computational models. This Agent-Oriented Software Engineering discipline is the focus of the next chapter.

3 Agent-Oriented Software Engineering

This chapter presents the main ideas behind Agent-Oriented Software Engineering (Section 3.1). It presents a summary and comparison of the most well-known agent-oriented software development methodologies (Section 3.2) and shows the most common steps of these (Section 3.3). The chapter ends with some concluding remarks (Section 3.4).

3.1 Introduction to AOSE

Since developing software is much more than just writing a program, the software engineering discipline has become increasingly important. This implies not only following certain development process but also following certain *development paradigm*. While the former deals with issues like Waterfall or Iterative & Incremental development processes, the latter deals with the way in which the software is conceived, including how the problem is tackled, requirements are analyzed, a solution is designed and finally how this design is implemented in a programming language, all of which can be summarized as the *approach* for (thinking and) developing software.

In the same way that applying the object-oriented paradigm to develop software has led to Object-Oriented Software Engineering (OOSE), applying the agent-oriented paradigm has led to Agent-Oriented Software Engineering (AOSE).

Even though this thesis emphasizes the adoption of an agent-oriented methodology for ultimately developing an agro-ecosystem simulation, AOSE can be seen as a promise for tackling the ever-growing complexities of software development in general. In many cases it is recognized that the behavior of a large-scale software system can be assimilated more appropriately to a human organization aimed at reaching a global organizational goal, or to a society in which the overall global behavior derives from self-interested intentional behavior of its individual members, than to a logical or mechanical system [Petra et al., 2003]. Some authors go further stating that it seems very likely that the software engineering of tomorrow for addressing more complex societal problems will be agent-oriented, such as that of today is object-oriented[6] [Perez and Batten, 2006].

3.2 Review of AOSE Methodologies

This section provides a summary of the ten most well-known agent-oriented software development methodologies, which are presented in [Henderson-Sellers and Giorgini, 2005][7] as well as a very broad comparison between them..

The following is a very brief summary of these ten methodologies:

- The **Tropos** methodology [Bresciani, Giorgini et al., 2004] provides guidance for the four major development phases of application development. One of its primary contributions is placing an emphasis on modeling goals and their relationship with the system's actors, tasks, and resources.
- **MAS-CommonKADS** [Iglesias et al., 1996] is based on both CommonKADS and object-oriented (OO)-based methodologies. This enables the developer to build agent based systems while leveraging the experience of pre-agent methodologies and employing familiar techniques and diagrams.
- The **PASSI** methodology [Cossentino, 2005] brings a particularly rich development lifecycle that spans initial requirements through deployment and, in addition, emphasizes the social model of agent-based systems.
- From an AI planning-execution perspective, **Prometheus** [Padgham & Winikoff, 2002] provides an especially rich goal-driven approach for its BDI-like agents. Its methodology is used today to develop systems on commercial BDI-based agent platforms, such as JACK or Agentis.

[6] "*I've always felt that the future of our field [computer science] is more to be found in what the AI [Artificial Intelligence] folks were trying to do, than in what the workaday programmers in academia and business concern themselves with.*" Prof. Alan Kay, OOP and GUI pioneer (URL: http://www.amazon.com/gp/blog/post/PLNKVUNBWIDAS9YQ)

[7] "*This book* [referring to Henderson-Sellers and Giorgini, 2005] *is the first to present and explore the ten most prominent methodologies for developing agent-based systems.*" James Odell.

- **Gaia** [Wooldridge, Jennings & Kinny, 2000] is one of the earliest agent methodologies and now reflects this experience in version two of its approach. Using the analogy of human based organizations, Gaia provides an approach that both a developer and a non-technical domain expert can understand—facilitating their interaction.
- **ADELFE** [Bernon, Gleizes, Picard & Glize, 2002] is a specialized methodology that emphasizes cooperative agents that self-organize and possibly result in emergent systems. More specifically, it addresses designing complex adaptive systems and anticipating emergence within its software agents.
- Resulting from a two-year, European-funded EURESCOM project, **MESSAGE** [Caire et al., 2001] extends existing object-oriented methodologies for agent-oriented applications. Chartered to address telecommunications applications, its resulting RUP-based approach also supports more general applications.
- The **INGENIAS** methodology [Pavón, Gomez-Sanz & Fuentes, 2005] supports a notation based on five metamodels that define the different views and concepts of a multi-agent system. Using metamodels provides flexibility for evolving the methodology and adopting changes to its notation.
- **RAP** [Taveter & Wagner, 2005] is concerned with distributed information systems (such as enterprise resource planning and supply-chain management systems) and places less emphasis on AI-based systems. The philosophy of the Model Driven Architecture (MDA) is adopted with the goal that executable software agents can be generated using RAP artifacts.
- **MaSE** [DeLoach, 1999] is a comprehensive methodology that has been used to develop systems ranging from heterogeneous database integration applications to biologically based, computer-virus immune systems and cooperative robotics systems. Its hybrid approach can be applied to multi-agent systems that involve implementations, such as distributed human and machine planning.

Figure 7 shows a schematic representation of the relations between object-oriented and agent-oriented methodologies, showing direct and indirect influences between them (simplified from [Henderson-Sellers and Giorgini, 2005]). The figure shows all ten methodologies plus others that also served as influence (represented in italic white font in the figure) but which will not be further discussed:

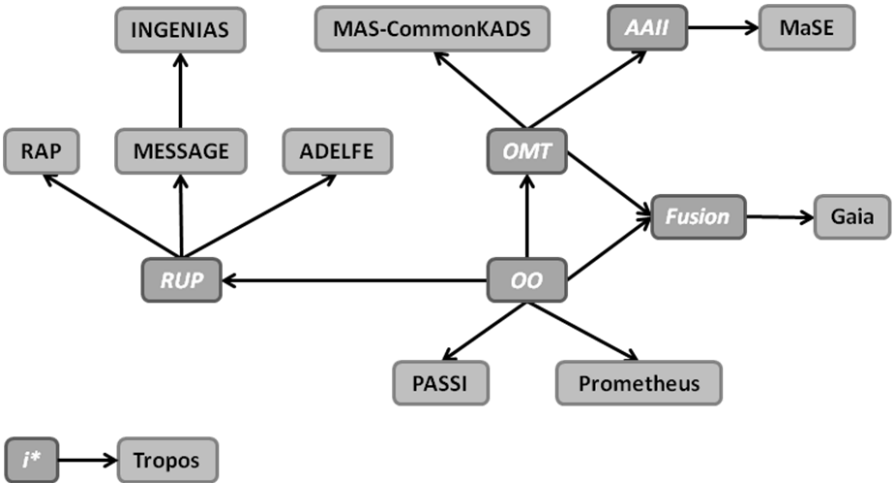


Figure 7: Influences between various OO and AO methodologies.

As it can be noticed in the figure, all ten methodologies (represented in bold above) except for Tropos, have been influenced by object-oriented techniques (represented as "***OO***" in the figure). Also several are reported to be inspired (according to [Henderson-Sellers and Giorgini, 2005]) on the Rational Unified Process (RUP) [Kruchten, 1999]. The Object Modeling Technique (OMT) [Rumbaugh et al., 1991] was one of the first OO methodologies which in turn influenced other methodologies like

the Australian Artificial Intelligence Institute methodology (AAII) [Kinny, Georgeff & Rao[8], 1996], and the OO methodology called Fusion [Coleman et al., 1994].

An interesting case is the one of Tropos[9], which is influenced by the "i* organizational modeling framework" [Yu, 1995]. This framework proposes an AOP approach to requirements centered on the intentional characteristic of agents. Even though the main concepts (actor, goal and dependency) are rather general, Tropos uses this framework to model early and late requirements, architectural design and detailed design for developing an agent-based software system.

Even though there are other methodologies for developing AOP-based software systems, this thesis is based on the ten presented in [Henderson-Sellers and Giorgini, 2005] since they are arguably the most referenced and cited elsewhere (for instance, in the International Workshop on AOSE [AOSE, 2010] that since 2008 is held in conjunction with the Journal of Autonomous Agents and Multi-Agent Systems [JAAMAS, 2010]).

The following table contains a brief comparison between these ten methodologies according to a certain criteria. The intention of this comparison is not to be exhaustive in the comparison (which is already done in Chapter 12 of [Henderson-Sellers and Giorgini, 2005]) but to point out some differences between them.

[8] Michael Georgeff and Anand Rao are pioneers in the research of BDI-like architectures, especially on the necessary logic behind them to represent beliefs, desires and intentions.
[9] Web-site: www.troposproject.org

Table 2: Comparing the ten AOSE methodologies (based on [Henderson-Sellers and Giorgini, 2005])

		Gaia	Tropos	MAS-Common KADS	Prometheus	Passi	Adelfe	Mase	Rap	Message	Ingenias
Process-Related Criteria	**Application domain**	*Independent*	*Independent*	*Independent*	*Independent*	*Independent*	*Dependent (adaptive systems)*	*Independent*	*Dependent (distributed organizational inf. systems)*	*Independent*	*Independent*
	Size of MAS	*<=100 agent classes*	*N/A*	*N/A (possibly any size)*	*Any size*	*N/A*	*N/A (possibly any size)*	*<=10 agent classes*	*Any size*	*N/A (possibly any size)*	*N/A (possibly any size)*
	Agent nature	*Heterog.*	*BDI-like*	*Heterog.*	*BDI-like*	*Heterog.*	*Adaptive*	*N/A (possibly Heterog.)*	*Reactive*	*Heterog.*	*Agents with goals and states*
	MAS approach	*OO*	*i**	*Knowledge Eng.*	*OO*	*OO*	*OO*	*OO*	*OO*	*OO*	*OO*
	Use of roles	*Yes*	*No*	*No*	*No*	*Yes*	*No*	*Yes*	*Yes*	*Yes*	*Yes*
Model-Related Criteria	**Autonomy**	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
	Adaptability	*Possibly*	*No*	*No*	*No*	*No*	*Yes*	*No*	*No*	*Possibly*	*Possibly*
	Cooperative	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
	Communication	*No*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
	Reactivity	*Possibly*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
	Persistence	*No*	*No*	*No*	*No*	*No*	*No*	*No*	*No*	*No*	*No*
	Deliberative	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*No*	*Yes*	*Yes*
Supportive-Feature Criteria	**SW support**	*No*	*No*	*No*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
	Open systems	*Yes*	*No*	*No*	*No*	*No*	*Yes*	*No*	*No*	*No*	*No*
	Dynamic structure	*No*	*No*	*No*	*No*	*Yes*	*No*	*Possibly*	*No*	*No*	*No*

Legend: N/A stands for 'information non available'.

The following is a description of the different criteria groups that were used in **Table 2**. This serves the better understanding of the table as well as it facilitates a deeper insight on these general-purpose AOSE methodologies, and the AOSE discipline itself.

Process-Related Criteria: These criteria look at the applicability of the methodology, the steps provided for its development process and the development approach followed:

- **Application domain**: Is the methodology applicable to any application domain (i.e. domain independent) or to a specific domain (i.e. domain specific)?
- **Size of MAS**: To what size of MAS is the methodology suited?
- **Agent nature**: Does the methodology supports, agents of any architecture, reasoning mechanism, and/or knowledge representation (i.e. heterogeneous agents), or only agents of a particular type (i.e. homogeneous agents).
- **Generic MAS development approach**: for instance OO approach or knowledge engineering approach.
- **Approach towards using "role" in MAS development**: Does the methodology employ the concept of "role" in MAS analysis and design?

Model-Related Criteria: These criteria evaluate the various aspects of a methodology's models and notational components:

- **Autonomy**: Can the models support and represent the autonomous feature of agents (i.e. the ability to act without direct intervention of humans or others and to control their own states and behavior)?
- **Adaptability**: Can the models support and represent the adaptability feature of agents (i.e. the ability to learn and improve with experience)?
- **Cooperative behavior**: Can the models support and represent the cooperative behavior of agents (i.e. the ability to work together with other agents to achieve a common goal)?
- **Communication ability**: Can the models support and represent knowledge-level communication ability (i.e. the ability to communicate with other agents with language resembling human-like speech acts)?
- **Reactivity**: Can the models support and represent the reactivity of agents (i.e. the ability to selectively sense and act in a timely manner)?
- **Persistence (temporal continuity)**: Can the models support and represent temporal continuity of agents (i.e. persistence of identity and state over long periods of time)?
- **Deliberative behavior**: Can the models support and represent the deliberative behavior of agents (i.e. the ability to decide in a deliberation or proactiveness)?

Supportive-Feature Criteria: These criteria assess the various high-level, supplementary features of an AOSE methodology:

- **Software and methodological support:** Is the methodology supported by tools and libraries?
- **Open systems:** Does the methodology provide support for open systems (i.e. that allow for dynamic addition/removal of agents)?
- **Dynamic structure**: Does the methodology provide support for dynamic structure (i.e. allows for dynamic reconfiguration of the system)?

From analyzing these ten AOSE methodologies it follows that none of them is aimed at simulating a MAS, and neither to be applied to agro-ecosystems. In a broader search, no AOSE methodology was found to be specially aimed at simulating agro-ecosystems. In an attempt to build such a methodology (or at least a methodological framework) the next sections will use those steps as a basis, that generally appear on these ten AOSE methodologies.

3.3 Standard Steps of an AOSE Methodology

An important finding in [Henderson-Sellers and Giorgini, 2005] is that there is no study that identifies the "standard" steps that can be followed in any AOSE development process. This "standard list of steps" would enable to have a common framework from which to compare the different methodologies (e.g. How does Met. A differs from Met. B regarding their process steps?) and could be a basis for building an AOSE methodological framework.

Based on this, Tran & Low proposed 19 "standard steps" and in order to verify their validity, they conducted a survey of experts who expressed their opinions on the importance of these steps. The survey confirmed their validity but unfortunately the authors gave no details about who these experts were.

It is understood that no single methodology will include all of these standard steps, but each will provide support to a subset of these, and furthermore, some will provide an explicit support for certain step while others may provide implicit or partial support (for a complete list comparing each of the ten methodologies with these 19 steps refer to Chapter 12 of [Henderson-Sellers and Giorgini, 2005]).

For the purpose of this thesis, such a list of standard steps allows to see the general activities shared among several methodologies, giving a more abstract view than the analysis each and every one of the ten methodologies, and may also allow to identify which of these steps are needed in order to successfully design and develop agro-ecosystem simulation's models.

From these original 19 general steps, Tran and Low recently distilled 16 general steps [Tran and Low, 2008] which are shown in **Table 3**. **Table 4** contains the particular to each methodology. The reason for including the latter is because one or more "specific steps" could be identified, which could be useful for developing agro-ecosystem simulations.

Even though the different general steps in **Table 3** are numbered and presented in that order, this doesn't mean that these steps should be performed (if at all) in that order. As said, these steps do not actually represent any particular AOSE methodology; they were abstracted from several AOSE methodologies.

	AOSE Step	Description
Problem domain analysis steps	**1.** Identify system functionality	Determine what the system should do in terms of functionalities (the same as traditional requirement engineering techniques).
	2. Identify roles	Identify the roles that should later be played by the different agents. This step may include analyzing the organizational context in which the MAS will be deployed since roles commonly (and naturally) appear in real organizational contexts.
	3. Identify agent classes	Identify which types of agents are needed (agent classes). This can be related to roles in the sense that a certain agent class can be created to cope with certain role characteristics.
	4. Model domain conceptualization	Identify the domain's main concepts and relationships. This step is frequently associated with developing the so-called "ontologies" in several methodologies, which are very similar to domain diagrams (conceptual modeling), easily depicted in UML Class Diagrams.
Agent interaction design steps	**5.** Specify acquaintances between agent classes	Determine basic relationships between agent classes (such as who knows who, possible hierarchies, etc.)
	6. Define interaction protocols	Define how agents can interact by defining communication protocols between them.
	7. Define content of exchanged messages	Define what the messages between agents will contain. These 3 steps (which comprise the "Agent Interaction Design") are very related and can be considered altogether.
Agent internal design	**8.** Specify agent architecture	Determine the specific (internal) architecture for each agent class (like for example BDI architecture). The specific architecture to choose will depend on the characteristics that the different agent classes should present (e.g. if they must conceptualize beliefs, goals and plans, or if

steps		they will be just reactive).
	9. Define agent mental attitudes	If the agent is to present mental attitudes like goals, beliefs, plans or commitments, define these for each agent class. This step includes trying to identify how these attitudes will be design internally for the agents' classes that need them.
	10. Define agent behavioral interface	Determine the capabilities, services, contracts and any other programming interface that the agent must provide. This will depend on the agent architecture (e.g. if it is a deliberative agent or if it is a reactive agent).
Overall system design steps	**11.** Specify system architecture	Overview of all components and their connections (in a higher level than Agent Architecture). This step also includes resources and environment (if they are of interest) and their relation to the agent classes.
	12. Specify organizational structure/inter-agent social relationships	If the system can be analyzed in terms of organizational concepts, then specify the organizational structure and the inter-agent social relationships. This step can be related to **Step 2** (Identify Roles) since roles are generally crucial in any organizational structure, as well as hierarchies between them. There are also several pre-defined organizational structures available (e.g. peer to peer, hierarchical, etc.).
	13. Model MAS environment	If the environment is relevant, then identify its resources, facilities and characteristics. Although part of this step can be done in **Step 12** (mainly those structural environmental aspects) others (mainly those functional aspects) may not.
	14. Specify agent-environment interaction mechanism	Closely related to the previous two steps, it involves detailing how the different agent classes are to be related to their environment, (analogous to interaction protocols, but instead of agent-agent interactions here we see agent-environment interactions). It is worth noting that this step can actually influence the architecture chosen for the different agent classes (this stresses the idea that these general steps are not necessarily presented in order).
	15. Instantiate agent classes	They determine how many agent instances will be needed for each agent class at initialization of a simulation.
	16. Specify agent instances deployment	Determine where the agent instances are to be (physically) deployed. This includes specifying agent platforms, nodes (with processing power), connection between nodes, etc. UML Deployment diagrams can be used.

Table 3: The 16 general steps identified by [Tran and Low, 2008] present in the major AOSE methodologies.

Step	Description
Specify organizational rules	Identify rules that must be respected by the organization for it to work coherently, and specify how the dynamics of the organization should evolve over time.
Identify initial sub-systems	Determine whether multiple organizations have to co-exist in the system and become autonomous interacting MASs
Implementation	Map the system design components to concrete components in the development framework and generate codes using code-generation tools.
Elaborate and validate UI prototypes	Specify the GUIs through which users will interact with the systems, and the relationships between GUIs.

Table 4: Other steps (not included in Table 3) that are only specific of certain AOSE methodologies.

3.4 Final Remarks

This chapter introduced the main concepts behind AOP and AOSE. It also presented the most well-known AOSE methodologies, none of which are specifically targeted to simulating a MAS, nor they are aimed at the domain of agro-ecosystems.

Since the objective of this thesis is to find, or build an AOSE methodology targeted at simulating an agro-ecosystem, and none was found to meet these requirements, the 16 general steps find by Tran and Low will serve as the basis for building such a methodology.

To this end, the following chapter aims at identifying, from these 16 general AOSE steps, those that would be useful in the context of this work, and adding others if necessary.

4 Towards a Methodological Framework for Developing Agro-ecosystem Simulations

The aim of this chapter is to build a methodological framework for developing agro-ecosystem simulation software based on the agent-based modeling approach. This problem could be divided into two parts. First, how to simulate a MAS, and second, how to simulate a MAS that represents an agro-ecosystem.

To this end, this chapter presents the general guidelines that will be considered for the methodological framework (Section 4.1), and then the requirements for simulating a MAS (Section 4.2). Based upon these requirements, the 16 general AOSE steps introduced in Chapter 3 are analyzed in order to select only those that cope with them (Section 4.3). Since new steps, besides these 16, may be needed, Section 4.4 discusses which new steps could be added for successfully simulate a MAS. Later, Chapter 5 presents the more specific agro-ecosystem requirements and how the methodological framework supports them.

The premise of this chapter is that none of the studied AOSE methodologies explicitly support the development of agro-ecosystem simulations. As introduced in Chapter 1, the full definition of a detailed methodology for developing agro-ecosystem simulations is out of the scope of this thesis.

4.1 Guidelines for the Proposed Methodological Framework

Even though the steps will be presented in a sequential order, the intended approach of the methodological framework is iterative and incremental. This means that any of the steps may be revisited several times, one per iteration, and each time giving a more complete solution.

Several options are available when simulating certain processes using software, like discrete-event, continuous and time-stepped simulations. The method by default for the methodological framework will be time-stepped.

By comparing the time-step scheduling and discrete-events approach [Galler, 1997] notes the advantages and disadvantages of each. He studied how different approaches can address the issue of simultaneous events and competitive activities, but also their respective performances in terms of computing time. Without concluding a preference for either, he notes that the continuous time approach does not provide clear benefit. Because the development and implementation of an event-based simulator are not obvious, many errors difficult to detect can occur. The author then concludes that for practical reasons, it is better to use a discrete time approach, which greatly simplifies the development of simulations, but the granularity of time must be accurately determined.

Also, the time-stepped approach has shown to be very natural when discussing with stakeholders (especially producers), since these tend to think of their activities and decision-making on a time-step fashion, like decisions taken every day, month, season or year.

There are also several different graphical modeling languages that could be used to express the different artifacts. Some of them are general-purpose, like UML [UML, 2010] and others are specific to MAS, like the Agent Modeling Language or AML [AML, 2010]. Since the aim is to facilitate developer's uptake so that they can use their preexisting knowledge of both, OOP and UML, the methodological framework will use plain UML diagrams. Nevertheless it is worth noting that using MAS-specific languages like AML allows for better specifications, since specific MAS constructions are available. Interestingly, AML itself was inspired by five of the ten general-purpose AOSE methodologies presented in [Henderson-Sellers and Giorgini, 2005] and discussed in Chapter 3, namely TROPOS, GAIA, PASSI, PROMETHEUS and MESSAGE[10], something that also stresses out the relevance of these methodologies.

In addition to the use of a graphical modeling language for specifying the various MAS constructs, developers also need a programming language for implementing them. Here some Agent-Oriented Programming Languages are also available, but in order to achieve the uptake objective already mentioned, the Object-Oriented Programming paradigm will be followed.

Regarding the software development process, an Iterative & Incremental process is encouraged. This means that certain steps may be revisited. Because it is out of the scope of this thesis to define a complete and comprehensive software development methodology, further references and

[10] AML specification can be found at: http://www.whitestein.com/library/whitestein_aml-specification_v09.pdf

details on how to follow this Iterative & Incremental approach will be omitted. Nevertheless, this implies that the proposed artifacts need not be developed in the sequential order in which they are presented here.

Since the aim is to develop an agro-ecosystem agent-based model for *simulation*, there are two ways to achieve this: either to develop the entire software or to use an already existing MAS simulation software package (also known as *MAS simulation frameworks*). These are general-purpose software packages that provide abstract classes and commonly used behaviors for the developer to use and extend, all in the context of MAS (that is, providing constructs at least for the concept of agent). Even though the steps of the proposed methodological framework will be useful for both ways, it is recommended to use, whenever possible, an already existing MAS simulation framework, since the workload may be significantly reduced. This is why the steps will be presented assuming that such a framework will be used.

Finally, the work of [Le Page and Bommel, 2005][11] also served as an inspiration for this thesis, given that they propose the use of UML as a means for specifying and CORMAS [CORMAS, 2010] as a means for implementing MAS simulations. Nevertheless, they present in a very general way how some UML Diagrams could be used to implement a CORMAS simulation, without providing further methodological aspects on how a software methodology could be followed (e.g. steps and artifacts to build on those steps). There is also no mapping given on how the most relevant features present in agro-ecosystems could be successfully represented using those diagrams; no details on how to model the simulation aspects (such as initial configuration, input and output parameters or visualization). In addition to this, the notion of role is not used, and they only provide a "toy-model" example instead of a real-world one.

In summary, the guidelines that will be taken into consideration for building the methodological framework will be the following:

- **Iterative & Incremental**: although not thoroughly presented in this thesis, the methodological framework will be assumed to be followed in an Iterative & Incremental way.
- **Time-step**: it will assume a time-step type of simulation, where the time-step duration should be defined and it will remain constant throughout each simulation run.
- **UML**: plain UML diagrams will be used.
- **OOP**: the OOP paradigm will be suggested for implementation.
- **Simulation framework**: the use of an already existing simulation framework software package is suggested, and this thesis' case study (Chapter 6) will make use of the CORMAS simulation platform as it is dedicated to natural resources management (i.e. to model and simulate agro-ecosystems).

4.2 MAS Simulation Requirements

In order to model and simulate an agro-ecosystem using the agent-based modeling approach, the following elements should be taken into account:

- **Agents**: they will naturally be at the core of any attempt to develop a simulation using an ABM approach. This also includes interactions between agents.
- **Environment**: it is what surrounds the agent, where the agent is located, and a means by which the agent receives input.
- **Resources**: they represent what the agents produce and/or consume. Even though this concept is not generally defined as a separate aspect of an ABM, it is particularly useful in the context of agro-ecosystems due to the importance of representing the production and/or consumption of resources and the relation between resources and the environment. In this context, resources are generally thought of as passive objects that are, for example, produced by the environment and consumed by agents within certain process (like crop production).
- **Simulation capabilities**: since the objective is to simulate agents, environment and resources, the methodological framework must allow this to happen. This includes (but is not limited to): the notion of time, the possibility to configure different initial situations from which to start the simulation, configuration of input parameters, and output results.

[11] Which is available at: http://cormas.cirad.fr/pdf/AsiaBook/20_LePage_2005_BC.pdf

All of these elements configure a good starting point since they give an overview of the requirements that must be met in order to develop an agro-ecosystem simulation. The next section takes these into account for selecting, from the 16 general AOSE steps of Table 3, those that meet these requirements.

4.3 Selecting Steps

This Section presents an analysis of each of the 16 general AOSE steps presented in **Table 3** in order to identify those which are suitable for developing a MAS simulation. That is, those that cope with the requirements of **Section 4.2**. For each general step, a discussion is presented to justify whether to select it or not for the methodological framework (under the title "*Discussion about considering this step*"). Furthermore, an analysis on what information would be manipulated in the step (under the title "*What information should be generated in this step?*"), and how that information could be best represented (by means of artifacts, usually models, under the title "*Possible artifacts of this step*").

Problem Domain Analysis Steps

General Step 1: Identify System Functionality

Discussion about considering this step
This step determines what the system (software) should do. It was thought as part of a general AOSE methodology, so as in any general-purpose software engineering methodology, it is vital to state what the software should do (e.g. register all the sales, including the customers that purchased them and perform queries over that data). On the other hand, MAS like those being studied in this thesis already have a special focus: MAS simulation, and more specifically, agro-ecosystem simulation. This greatly reduces the scope of what the system should do, although the software developer must state, within that specific scope, what the system should simulate, and more importantly, the objective of the simulation. This is why this step will be considered, but also renamed as "*Identify System Purpose*".

What information should be generated in this step?
This step should define the purpose of the simulation (its objective), as well as provide an overview of the context in which the simulation will be developed (e.g. what would be the use of it, who would use it and develop it, who would be represented in it, why the simulation would be developed and what would be expected from it). This very much resembles the first documents generated in general-purpose software engineering methodologies, but it should contain some other information not generally present in these, namely to clearly explicit the objective of the simulation's model, and the people involved, either being modeled and represented inside the simulation as well as those that will be the modelers and developers. The reason for adding this information is that any model will invariably include the modeler's perspective or point of view. This may not be relevant if the subject being modeled is an invoice or a product purchase, but agro-ecosystems will represent human behavior as well as real-world problems and conflicts, and depending on how the modeler models this, the result of the simulation may change. Depending on sensitivity issues, even small differences on perspectives between modelers that affect their models may have a tremendous impact on simulation output. Like [Grimm & Railsback, 2005] point out, clarifying the purpose of the model is essential because it provides a framework for its whole description.

Possible artifacts of this step
Since this step doesn't include any modeling, a text document should be enough. This document will include (but not be limited to) the following items:

- **General system purpose**: representing the general objective and giving an overview of why the simulation system is to be built.
- **Questions that the system should answer**: representing specific objectives.
- **Stakeholders**: including anybody that has an interest, either direct or indirect on the system that will be developed, or on the results that could be obtained.
- **Target population**: for which stakeholders the system will be developed, differentiating those that will be modeled *into* the system and those that will *use* the system.
- **Modelers**: who will model and develop the system, in order to help identifying any risk of adding a perspective or bias.

General Step 2: Identify Roles and

General Step 3: Identify Agent Classes

Discussion about considering these steps

A role is the functional or social part which an agent, embedded in a multi-agent environment, plays in a process like problem solving, planning or learning [Lind, 2008]. The role defines what the agent is expected to do within certain organization and considering other agents. In the context at hand, the system will probably present several different roles, like farmer, consumer, local authority, etc. These are roles people play in a society and in an agro-ecosystem. Roles help determine agent types (or 'agent classes' as called in the general AOSE **Step 3**). Moreover, agent classes can be viewed as specializations of roles, more concerned about agents than about social behaviors (as roles do). An important issue about roles is that any given agent can play multiple roles, either simultaneously or in turns. This may lead to define agent classes that take this into account (e.g. an agent class 'farmer authority' that merges two roles: farmer and local authority). Since these concepts may be crucial to model and simulate agro-ecosystems, and are related to each other, these steps will be considered together and renamed as "*Identify Roles and Agent Types*".

What information should be generated in this step?

This merged step involves specifying the expected behavior of the role within the system and with respect to other roles (hence to other agents). Since roles are a functional or social part and define what the agent playing that role is expected to do, then a comprehensive description of a role must state the expected behavior of any agent that will play that role.

Since the methodological framework is intended to have an iterative and incremental approach, this step could only identify, at first, roles and agent types (e.g. assigning them a name and overview), and only later on be revisited when the modeler/developer has enough information to fully describe each role and agent type.

Possible artifacts of this step

There are several artifacts that could be built since different information is needed:

- **Roles**: text document including the names and general description for each role. The description should include the goals of each role as well as a general overview of the responsibilities related to that role (that agent types playing that role must present) as well as relations with other roles (if they apply).
- **Agent Types**: text document identifying the names of the different kinds of agents that will be modeled, their goals, objectives and relation with the roles identified. This identification should be consistent with the ***Domain Model Conceptualization Step*** since it represents the structure of the system, which includes the different kinds of agents (e.g. if three different types of agents are identified, then those three kinds should appear as domain concepts).
- **Agent Types' Behavior**: a thorough description of each agent types' behavior. This behavior must be specified for a certain moment in time, representing those decisions that the agent type will make at that time (with the information available at that moment). This means that this decision-making will be executed during the simulation at every time step (e.g. describe the logic of decision-making of local authorities at the beginning of the year, assuming a yearly time step). Even though these behavioral descriptions could be documented by means of a textual description, UML Activity Diagrams can be used to complement such descriptions.

General Step 4: Model Domain Conceptualization

Discussion about considering this step

This step looks for the domain's main concepts and relationships. It is a static (structural) view of reality, in contrast with roles and behaviors which are a dynamic (behavioral) view. This is why they complement each other and they must also be consistent with each other. This step has exactly the same purpose than domain models of traditional OOSE methodologies: to identify those concepts that are most relevant for the domain being studied and that will be of interest for the system to be built (in this case, for the simulation that will be developed). Since it is not possible to automatically identify those most relevant concepts, the modeler must get involved with the domain and determine which

elements, from the nearly infinite number of elements that reality is composed of, will be part of the domain model. This step will be considered, using its original name "*Domain Model Conceptualization*".

What information should be generated in this step?
Since only one domain model conceptualization will be built, it must include not only the agent types but also the most relevant elements that compose the environment as well as resources. All of this must be modeled by their structural nature, since behaviors are not to be included in this model. Based on **Section 4.2**, the domain model (applied to agro-ecosystems) will group three kinds of concepts: those related to agents, those related to the environment and those related to resources.

Possible artifacts of this step
The most well-known artifact for domain models is UML Static Structure Diagrams, also known as Class Diagrams. As stated before, the methodological framework is based on the use of plain UML.

Agent Interaction Design Steps

General Step 5: Specify Acquaintances between Agent Classes

Discussion about considering this step
The aim of this step is to identify which other agent classes each one knows. This was already done, either when identifying agent types and roles, and/or when modeling the domain conceptualization, by means of associations between concepts, hence this step will not be considered by itself but as part of previous steps.

General Step 6: Define Interaction Protocols and

General Step 7: Define Content of Exchanged Messages

Discussion about considering these steps
Communication between individual agents (each being an instance of an agent type) must be allowed, and will depend on how each agent type interacts with the others (e.g. for letting agent instance 'a' of type A communicate with 'b' of type B, types A and B must know each other). Because their very nature agents can be autonomous, it is not expectable that they already know how to establish a communication with each other (in contrast with common OOP objects that already know each other's interfaces beforehand). That's why this communication involves determining some common interaction protocol as well as defining all possible messages that could be exchanged. Nevertheless, in the case of developing an agro-ecosystem simulation, agent types will be developed altogether, so the autonomy and independence of individual agents is much reduced compared to other MAS scenarios. This facilitates the task of communication between agents, and enables considering these two steps as one, renamed as "*Define Agent Interaction*".

What information should be generated in this step?
This step should determine when, how and what the different agents will communicate. *When* implies stating what triggers the communication, *how* implies defining the interaction protocol that agents will follow, and *what* implies knowing the content of the exchanged messages.

Possible artifacts of this step
There are specific agent-interaction diagrams such as those proposed by the Foundation for Intelligent Physical Agents or FIPA [FIPA, 2010], which allow for accurate and domain-specific representations of interactions in general-purpose MAS scenarios. However, considering on one hand (as previously discussed) that in the case of agro-ecosystem simulations agent's autonomy is much reduced, and on the other hand remembering one of the aims of the proposed methodological framework of helping developers uptake the methodological framework by using languages they already know, this step will use UML Sequence Diagrams in order to specify these interactions.

Agent Internal Design Steps

General Step 8: Specify Agent Architecture

Discussion about considering this step
This step involves specifying the internal architecture of each agent type. As **Section 2.3.2** introduced, there are already-defined general agent architectures, namely *deliberative* (that manage explicit representations of goals, beliefs, actions and use abstract reasoning tools), *reactive* (where agents simply react with an action triggered by some stimuli) and *hybrid* (that include a deliberative as well as a reactive component). Every agent type must present one of these architectures and since the software design will become much more complex if a deliberative architecture is chosen (and much simpler if a reactive one is) this step will be considered, although also including internal agent type class design. That is, after deciding the agent type architecture, design the internal agent type structure and behavior that supports and details that architecture. This step will be renamed as "*Agent Architecture and Design*".

What information should be generated in this step?
This step should generate enough information for detailing the agent type's internal class design, following the chosen agent architecture. In general terms this means describing how the agent will be able to follow the Perceive/Reason/Act cycle presented in **Figure 4**. This step is similar to the Architecture and Design steps of an OOSE methodology where the aim is to first decide over an architectural style (e.g. Layered) and then define the necessary classes that will populate that style.

Possible artifacts of this step
UML Class Diagram for specifying each agent type architecture as well as their internal design. In order to represent internal design behavior (functional aspects) UML Behavioral Diagrams may also be used if necessary (e.g. Communication, Activity or State-Transition Diagrams).

General Step 9: Define Agent Mental Attitudes

Discussion about considering this step
Mental attitudes should be involved if a deliberative (or hybrid) architecture is chosen, so this step will be considered as part of the previous step (if needed).

General Step 10: Define Agent Behavioral Interface

Discussion about considering this step
Originally this step aimed at determining the agent's capabilities and services it will provide. Since this thesis focuses on agro-ecosystems simulations, the only capabilities and services an agent will present are those already considered in its role, agent type description, domain model concepts and agent interaction. The agents of the agro-ecosystem will not provide services to other software entities (as agents in other context may do) so the "services" they have are actually the messages they will exchange between them (which were already considered in the "*Define Agent Interaction*" step). In summary, what should happen is that this step should not add anything not already considered by other steps, since it was originally thought for more general purpose MAS. This is why this step will not be considered.

Overall System Design Steps

General Step 11: Specify System Architecture

Discussion about considering this step
The original idea for this step was to give an overview of all components and their connections in the highest possible level. Once again, since the context is agro-ecosystems, this high level overview is already given (agents interacting with other agents as well as with their environment and resources). Another useful situation where the architectural view may be of assistance is when having physically distributed components, which is not the case for the agro-ecosystem simulations considered in this thesis. This step will therefore not be considered.

General Step 12: Specify Organizational Structure/Inter-Agent Social Relationships

Discussion about considering this step
Even though agro-ecosystems may involve some kind of organizational structure (like farmers organizations) they are not driven by any organizational metaphor as this step assumes. Also, no inter-agent social relationship was defined in "*Identify Roles and Agent Types*" or in "*Define Agent Interaction*", so this step will not be considered.

General Step 13: Model MAS Environment

Discussion about considering this step
The environment is indeed one of the crucial elements to consider, as **Section 4.2** pointed out. It could already be modeled in "*Model Domain Conceptualization*" at least from a structural point of view. Nevertheless, other aspects may also be need consideration, like those concerning a functional point of view for both, the environment and the resources it may provide. This is why this step will be included, although renamed, so as to include resources: "*Model Environment and Resources*".

What information should be generated in this step?
This step could, first, give a more detailed structural description of the environment and/or resources that may have not been included in the Domain Model (e.g. because generally domain models are very early developed, and in the Iterative & Incremental approach, more details about the environment and resources may appear afterwards and may introduce too much detail to the domain model). Second, behavioral aspects of the environment and resources where yet never specified (e.g. the behavior of a certain resource over time, or how it interacts with other resources and with the environment).

Possible artifacts of this step
UML Class Diagrams will allow for that detailed structural specification and UML behavioral diagrams will allow for specifying those behavioral aspects of the environment and resources that were not yet captured. An example of specifying the behavior of a resource could be to develop a UML Activity Diagram, much like those developed for agents, that determines how the resource behaves at each time step (e.g. how does certain crop grows over time). An example of environment behavior could be to develop a State-Transition Diagram for modeling the different stages of grass over time.

General Step 14: Specify Agent-Environment Interaction Mechanism

Discussion about considering this step
Even though it is possible to determine the interaction between agents and the environment as a separate step (as this general step suggests) it is just another kind of agent interaction. Considering them as separate steps (agent-agent interaction in one step, and agent-environment interaction on another step) may complicate the simulation and it would give benefits only for the case where each of these interactions is complex. Therefore these two steps will be performed simultaneously. This means to specify agent-environment interaction at the same time with agent-agent interaction. This is why this step will not be considered by itself but as part of either "*Identify Roles and Agent Types*" or "*Define Agent Interaction*".

General Step 15: Instantiate Agent Classes

Discussion about considering this step
This step involves determining how many agent instances will be needed for each agent type . This is indeed necessary for agro-ecosystem simulations but as part of the initial configuration of the simulation, which will evolve over time. There are also a number of other elements to initialize, so this step will not be considered by itself but as part of a new step that will come from simulation requirements (later discussed in **Section 4.4**).

General Step 16: Specify Agent Instances Deployment

Discussion about considering this step
Since the entire agro-ecosystem simulation will be run in only one computer, there is no need to consider physical distribution or deployment. Therefore this step will not be considered.

Table 5 summarizes the results of this section:

	General AOSE Steps	Selected Steps	Artifacts for Selected Steps
Problem domain analysis steps	**1.** Identify system functionality	**1. Identify System Purpose**	**Text Document**: including system objective, questions that the system should answer, identification of stakeholders, target population, and modelers.
	2. Identify roles	**2. Identify Roles and Agent Types**	**Text Document**: for role's identification and description. **Text Document**: for identifying agent types and their relation to roles. **UML Activity Diagrams**: for agent type's behavior specification
	3. Identify agent classes		
	4. Model domain conceptualization	**3. Model Domain Conceptualization**	**UML Class Diagram**: for modeling the domain's main concepts.
Agent interaction design steps	**5.** Specify acquaintances between agent classes	*No. This is either included in "2. Identify Roles and Agent Types" or in "3. Model Domain Conceptualization"*	
	6. Define interaction protocols	**4. Define Agent Interaction**	**UML Sequence Diagrams**: for representing the interactions between agents and between agents and the environment.
	7. Define content of exchanged messages		
Agent internal design steps	**8.** Specify agent architecture	**5. Agent Architecture and Design**	**UML Class Diagram**: for architecting and designing the internal structure of each agent type. **UML Behavioral Diagrams**: for specifying internal communications between objects (if necessary)
	9. Define agent mental attitudes	*No. If mental attitudes are to be considered for certain agent type (e.g. because a deliberative architecture is chosen) then these attitudes will be defined in the previous step.*	
	10. Define agent behavioral interface	*No. Interfaces between agents and with the environment will be the only ones and are specified in "4. Define Agent Interaction" and "6. Model Environment and Resources" respectively.*	
Overall system design steps	**11.** Specify system architecture	*No. The overview of components and connections is already given by the context of agro-ecosystem simulations.*	
	12. Specify organizational structure/inter-agent social relationships	*No. Agro-ecosystem simulations are not driven by organizational metaphors. This step is either included in "2. Identify Roles and Agent Types" or in "4. Define Agent Interaction".*	
	13. Model MAS environment	**6. Model Environment and Resources**	**UML Class Diagram**: for further modeling structural aspects of the environment and resources. **UML Behavioral Diagrams**: for modeling functional (behavioral) aspects of the environment and resources.
	14. Specify agent-environment interaction mechanism	*No. This step is either included in "2. Identify Roles and Agent Types" or in "4. Define Agent Interaction".*	
	15. Instantiate agent classes	*No. This step is part of the initial configuration of the simulation.*	
	16. Specify agent instances deployment	*No. There is no interest in physical distribution of agent types.*	

Even though this new set of steps covers the overall requirements stated in **Section 4.2**, there is still one that has not been met: simulation capabilities. This stated that agents, resources and environment should be placed into simulation runs, and this would imply such things as determining the time step of the simulation, what to do at every time step, the initial state of the system, input parameters that feed each simulation step, and output visualization.

The following section proposes which new steps could be added to the methodological framework in order to take the simulation requirements into account.

4.4 New Steps

This section introduces which steps could be added to the previous selected steps in order to have a methodological framework for developing agro-ecosystem simulations. For each new proposed step, a brief description will be given as well as the information that this step should generate and possible artifacts, similarly to **Section 4.3**.

New Step: Simulation Configuration

Description of this step

The aim of this step is to define those fundamental elements that will enable a simulation to be run. Since the simulation will be done by a simulation framework, it will need precise definitions for the initial state, input parameters, time step, etc. It is important to note that in order to properly perform this step, many elements should already have been defined (e.g. it would be impossible to state how the initial configuration would look like if no agents were yet defined). Also, besides the need for certain preceding elements, some of the information in this step may already be defined, like the case of the Time Step Definition or Task Scheduling. This may be because that information could have been needed for making decisions earlier in the development, and as stated in **Section 4.1** the methodological framework proposes an Iterative & Incremental approach. If that is the case, then that already defined information must only be recorded here.

What information should be generated in this step?

This step should define the following information:

- **Initial Configuration**: a simulation needs some starting point over which all the already-developed models and artifacts will act upon. This is called the *initial configuration* and determines the state of the simulation at time zero. The main elements to be determined are the initial number of agent instances (for each agent type), the distribution of these over the environment, the initial volume of resources and their location, among others. This information heavily depends on the actual reality (domain) being simulated.
- **Time Step Definition**: states how much time is considered to pass between two time steps (so called "step granularity"). The state of the simulation will change only at these time steps. Since these simulations are time-stepped the real-world meaning of passing one time step must be determined. This may impact on, for example, the decision-making of agents since agent behavior will probably not be the same if a time step represents one day or one year.
- **Task Scheduling**: in previous steps several elements (e.g. agents and resources) were assigned certain behavior (e.g. for decision-making and for evolution, respectively). This behavior is supposed to be called upon the simulation at every time step, in order to give each individual element (e.g. agent instance or resources instance) the chance to do something. What remains yet undetermined is the relative order in which these calls should be made. That is: at each time step the simulation should call agents to decide what to do and resources to evolve, but in what order? Is it the same to first have the agents make decisions and then let the resources to evolve? Or the other way around? And if the first scheduling is taken (first the agents and then the resources), are the agents randomly chosen to act or the order between them is important? Task scheduling means defining the order in which actions will occur at every time step of the simulation.
- **Input Parameters**: input parameters can be divided into two groups: static or dynamic. Static input parameters are those that will be fixed for the entire simulation period, so they represent data that is external to the various elements being modeled, although this is

assumed to remain constant over time (e.g. an interest rate). Dynamic input parameters are those that can vary within the simulation period, so they represent data that is external but that may change over time (e.g. seed prices varying in each season). Within the latter, two more subgroups can be defined: those that vary independently from the simulation results (e.g. international oil price that is independent from agent-decision at local scale[12]) and those whose value is (partially) determined by certain simulation results (e.g. certain crop whose price depends –among other factors- on the amount of product being produced in the simulation run). These possibilities finally lead to input parameters that can have the shape of constant values (e.g. 90) or functions (e.g. wheat_price(t)). Another aspect that is relevant to input parameters is where to take the values from. Two possibilities can be considered: a) to take historical values from the real world and b) to generate random values. Finally, it is important to note that input parameters are the main way to represent external elements of the system being modeled, and since no system can be completely isolated from its surroundings, input parameters may play a crucial role.

Possible artifacts of this step
Even though this step concerns several pieces of information which are crucial for the simulation, all of these will appear directly on the code, or if necessary may be properly described in text documents without the need of any special representation. Henceforth, further details about artifacts are not necessary. The only information that may take advantage of a tool such as UML is Task Scheduling. UML Sequence Diagrams capture the behavior of certain scenario [Fowler, 2003], so they are particularly useful for specifying the order in which things happen at every time step.

New Step: Simulation Output

Description of this step
Up to this point there is no way the simulation may answer any of the questions stated in the first step, therefore it will fail to achieve its objective. The reason is that there is no output coming out of the simulation. Thus, this step involves defining the output that the simulation should yield, as well as the way in which it should be visualized by users.

What information should be generated in this step?
This step should define the following information:

- **Output**: at this point, this output could be viewed as the analogous to the input parameters, in order to monitor what is actually happening inside the simulation, to see how certain key aspects are evolving (e.g. how many agents are dying or how much of a certain resource is left at every time step).
- **Visualization**: in relation to the output it is very useful to have some way of visually presenting results, values, and outputs to the user that is running the simulation (e.g. graphs). Besides output parameters, the environment can be shown in two or three-dimensional grids along with resources and agents. For example, resource levels can be graphically represented by a color scheme (red indicating low levels and green high levels), and agents can be viewed as they move along the cells of the grid. The way in which the simulation will show its results and current state is related to the user that will be running it: the more technical the user, the less visual representations he/she may need to correctly understand (and interpret) the results. However, the less technical the user, the more relevant the visual representations become.

Possible artifacts of this step
Similarly to what happened in the previous step, there is not much need for extra artifacts other than the source code itself (usually already done by the simulation framework) and, if necessary, explanatory text documents.

Figure 8 summarizes several of the concepts introduced in the previous two steps. It shows the change in the state of the simulation and which concepts influence it:

[12] These are also called "forcing variables" or "time-series input variables".

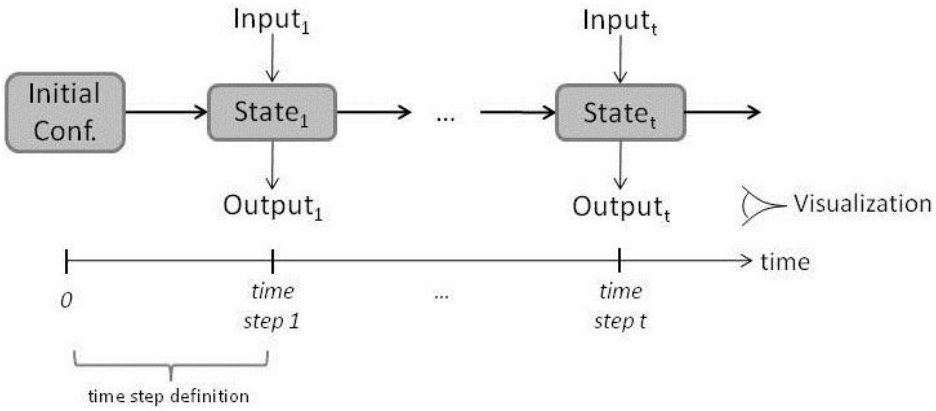


Figure 8: Schematic representation of the simulation concepts involved in a simulation run.

The previous (eight) steps build up the methodological framework for developing agro-ecosystem simulations. The following (two) steps are not the focus of this thesis and their introduction is merely for the sake of putting the previous ones in the context hence their detailed description is beyond the scope of this thesis.

New Step: Implementation

After completing each and every one of the previous steps, there should be enough information and models in order to start their implementation into a programming language. Either using an already existing simulation framework software package (as it is recommended in this thesis in order to write less code), or starting coding from zero without the help of these frameworks, the developer should have enough elements (as well as enough understanding of the problem) to start coding.

New Step: Simulation Run and Sensitivity Analysis

This step involves gathering all the necessary data for allowing the simulation to be run and to perform some explorations about how the output of the simulation is affected when certain elements are changed. This is known as sensitivity analysis[13].

Data gathering is related to input parameters, since these may need historical real-world data for dynamic parameters (e.g. evolution of certain price over time taken from real-world databases) or the definition of single-value static parameters (e.g. determine the price of oil that will be used for the entire simulation). This may also involve importing large amounts of data from spreadsheets or databases, so these aspects should be considered carefully, not only because of the possible technical difficulties that may arise[14], but also because a correct definition of input values determines the output of the simulation.

Sensitivity analysis aims at showing how the outputs of the simulation are susceptible to its inputs, or if a more white-box approach is made, susceptible to the decisions the developer makes (e.g. in the form of UML models).

If a black-box approach is taken, simulation outputs will only be analyzed for sensitivity against input parameters. This means, for example, how does certain resource level (like grass height) depends on certain input parameter (like sun radiation). These analysis help not only in finding potential errors but also to better understand the relationships between inputs and outputs, and considering the importance of relationships in complex systems, this becomes particularly relevant. Several methods exist for analyzing the sensitivity of certain output parameters in terms of several input parameters, either by changing one input parameter at a time (which assumes no

[13] "*New modelers often see building a model as their main task, but analyzing a model is every bit as essential as building it […]. Analyzing a model means doing research on a model to learn about its behavior and to learn about the system the model represents. Objectives of this research typically include verifying that the software does what we want it to, finding good model structures and theory for individual traits, finding good parameter values, and finally solving the problems we design the model for in the first place and learning something about ecosystems.*" [Grimm & Railsback, 2005].

[14] Like obtaining data from Microsoft Excel files or connecting to certain DBMS (database management system).

interrelationship between them) or by simultaneously changing several. Further details are out of the scope of this thesis.

If a white-box approach is taken, simulation outputs will be analyzed against the very internal decisions the software is making. For example: how does the production of certain resource (e.g. volume of wheat harvested) depends on certain agent's decision (e.g. either apply fertilizers or not)? This may lead to analyzing the output of the simulation considering the agent's decision (e.g. fertilize) and then without considering it (e.g. not fertilize). If this decision was already built in, for example, the UML Activity Diagram that represents the agent's decision making, then this diagram should be changed in order to analyze its impact on the output parameter. Since this involves knowing how the simulation works internally (and changing it) it is considered as a white-box approach.

Finally, both (input parameters, data-gathering and sensitivity analysis) strongly depend on the objectives of the simulation and on the questions it is supposed to answer. If for example, one of the key questions is "Examine the relationship between oil prices and producer's debt", then the focus of data gathering should be obtaining precise real-world historical values of oil prices (or thoughtfully determine some way of estimating it for the future) and the focus of sensitivity analysis should be in changing oil prices and exploring its impact on producers going in debt (black-box approach), and possibly also analyzing *how* and *when* producers go in debt (white-box approach).

4.5 Final Remarks

Table 6 summarizes all the steps of the proposed methodological framework discussed in this chapter.

These ten steps, six of which were taken from the 16 general-purpose AOSE steps (**Section 4.3**) and four added later (**Section 4.4**), have the overall aim of assisting in developing software that represents a MAS simulation, whose requirements where established in **Section 4.2**.

Since agro-ecosystems are a kind of complex system (**Section 2.2.2**) and these can be modeled using the ABM approach (**Section 2.4**), it naturally follows that an agro-ecosystem can be modeled and simulated using an ABM approach, and particularly following the methodological framework introduced in this chapter.

Nevertheless, there are quite specific agro-ecosystem features that are worth analyzing, such as how they could be represented, modeled and simulated by this methodological framework. This is the aim of the following chapter.

Proposed Step	Aim of the Step	Artifacts
1. Identify System Purpose	Define the purpose of the simulation, including its objective and questions to be answered.	Text Document including the purpose of the simulation, an overview of the context in which the simulation will be developed, including why the simulation will be developed and what will be expected from it.
2. Identify Roles and Agent Types	Identify agent types and roles, especially agent behavior.	Text Documents for role's identification and description and for identifying agent types and their relation to roles; UML Activity Diagrams for agent type's behavior specification.
3. Model Domain Conceptualization	Depict the structure of the problem, including entities and relationships.	UML Class Diagram for modeling main concepts, including those mentioned in steps 2 and 6.
4. Define Agent Interaction	Determine when, how and what the different agents will communicate.	UML Sequence Diagrams for modeling interactions.
5. Agent Architecture and Design	Define internal agent design (structure and behavior) in order to fulfill its perceive/reason/act cycle, within a simulation framework software package.	UML Class Diagram for designing the internal structure of each agent type; and UML Behavioral Diagrams for designing the internal behavior of each agent type.
6. Model Environment and Resources	Determine behavioral aspects (evolution) of resources and environment, and completing structural aspects.	UML Class Diagrams for further modeling structural aspects of the environment and resources; and UML Behavioral Diagrams for modeling functional (behavioral) aspects of the environment and resources.
7. Simulation Configuration	Define those fundamental elements that will enable a simulation to be run.	Source Code and explanatory text documents (if needed) for documenting the configuration and UML Sequence Diagram for tasks scheduling.
8. Simulation Output	Define the output that the simulation will produce along with its visual representation.	Source Code and explanatory text documents (if needed).
9. Implementation	Codify the simulation.	Source Code.
10. Simulation Run & Sensitivity Anal.	Answer the simulation's objectives and questions.	Text Documents with the conclusions.

Table 6: Summary of the steps that compose the proposed methodological framework.

5 Agro-ecosystems' Features Supported by the Proposed Methodological Framework

This chapter presents those features that characterize an agro-ecosystem and shows how they are supported by the methodological framework proposed in the previous chapter (see **Figure 9**). The objective is to show that this methodological framework can successfully represent them with the available steps and artifacts, leaving no relevant agro-ecosystem feature uncovered. The features presented were introduced in **Section 2.2.1**.

Some of the features that are presented in this chapter are often outside the boundaries of an agro-ecosystem. Nevertheless they are relevant because they affect it in some way that is of interest, but since they do not belong to the agro-ecosystem they will not be explicitly modeled (i.e. there will not be artifacts representing them). They will be considered mainly as restrictions in the inputs to the agro-ecosystem (more details are given in their corresponding sections within this chapter).

First, natural, human, capital and production resources-related features are shown to be supported (**Sections 5.1 to 5.4**). Afterwards other features are mapped into various steps and artifacts of the proposed methodological framework (**Section 5.5**). Finally, this chapter ends with some concluding remarks (**Section 5.6**).

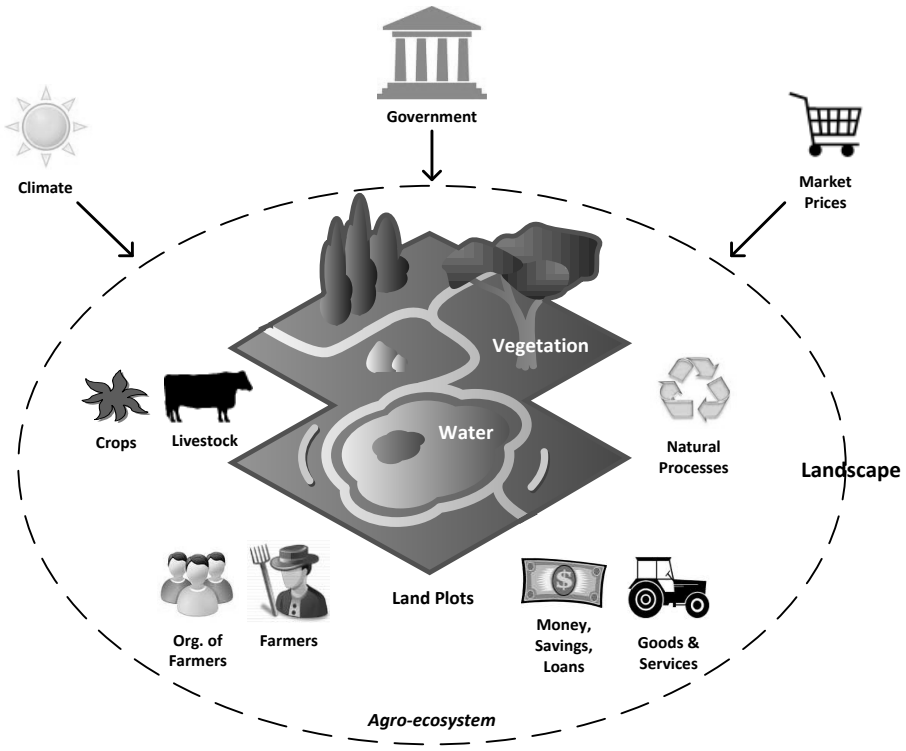


Figure 9: Schematic representation of relevant agro-ecosystem features.

5.1 Natural Resources Related Features

Natural resources are the given elements of land, soil fertility, water, climate and vegetation that are exploited by the farmer. All of these constitute crucial features that agro-ecosystems present that must be supported. The following sections first introduce the feature and then show how it can be mapped into the methodological framework in terms of its steps and artifacts.

Feature: Land

The space over which farmers live, crops grow and animals feed. Land can be divided into plots, and vicinity may be important. It can be modeled either into two or three dimensions, mainly depending on the topography (slopes, geographical accidents, elevations, etc.), and its relevance in the system under study.

Mapping to Methodological Framework

This feature is fundamentally supported by the concept of cellular automata (introduced in **Appendix A)** so it is an inherent part of the MAS approach. Also the land, along with water and climate, are natural candidates to be considered as components of the environment as introduced in **Section 4.2** ("environment is what surrounds the agent, where the agent is located"). Regarding the methodological framework, the step "*Model Domain Conceptualization*" proposes to identify those concepts that are most relevant for the domain and explicitly addresses the case of the environment. The Domain Model can represent 'plots' which can be thought of as land components, providing more flexibility by decomposing the land in several elements. About each plot, it can model its location (e.g. by coordinates), its vicinity (e.g. by associations with neighboring plots) as well as the relation between agents and plots (e.g. representing ownership) and between plots and resources (e.g. representing which resources are hosted in which plots). Plots allow for an easy geographical representation of land, where each plot is represented by a cell in a grid that can be shown to the user. Even depending on the objective of the simulation and the capabilities of the simulation framework software package being used, plots can be related to GIS (Geographical Information Systems) in order to visualize a real-world map instead of an abstract grid. Also depending on these capabilities, 2D and 3D grids of cells can be manipulated.

Feature: Water

Any form of water is generally so important to the agro-ecosystem that representing it is often necessary. It may be a river, a lake, or the ground water, but also the "water reservoir" due to the small pores of the soil that serve at supplying water to plants and other organisms during the rain-less period. The geographical relation to the land can also be of interest (e.g. how land plots determine access to water resources).

Mapping to Methodological Framework

Water can be represented as cells in a grid, so the problem is reduced to map a special kind of plot, which was discussed above. For more sophisticated models, water may be a soil property or even a water reservoir entity with its own complex dynamics.

Feature: Climate

Since weather directly affects an agro-ecosystem, elements such as rain, temperature, moisture or climatic events may be considered.

Mapping to Methodological Framework

Even though important, climate is not to be modeled within an agro-ecosystem simulation, but is to be considered as an input. The main reason for not modeling climate is simply because it is out of the agro-ecosystem limits. The first step added called "*Simulation Configuration*" allows to define which input parameters are to be used during each time step of the simulation, so it should consider climate information in it.

Feature: Native Vegetation

This is usually native vegetation already present in the land. As crop vegetation (cocoa trees, pasture, corn, etc.) is a key element of agro-ecosystems, it is described separately (see Section 5.4).

Mapping to Methodological Framework

Native vegetation can be represented jointly with land plots since it will grow over them. It is worth noting that crops are not considered vegetation but production resources.

5.2 Human Resources Related Features

As introduced in **Section 2.2.1**, human resources are the people who live and work within the farm and use its resources for agricultural production. This can be further analyzed by considering not only farmers which are the primary human resource, but also by considering different kinds of farmers (and leveraging the MAS approach that enables heterogeneous agents) as well as organizations of farmers or manpower labor forces.

Feature: Farmers

Since humans are what determine an agro-ecosystem, they must be represented. Even when different kinds of actors can be present, farmers are the ones that will always be present. Others (such as local authorities) can also be considered if it is important to include them as part of the system or, on the contrary, they can be considered out of the boundaries of the system and be indirectly represented by, for example, constraints on certain system elements (such as water availability because of local authority regulation). What should be modeled is the farmer's behavior, which makes them what they are. Labor availability is often another important factor related to human resources (e.g. how many healthy adults compose each family and their working capacity, for example, in hours per week). Even when considering only one kind of actor (farmers) it is rarely the case that they are so homogenous that can be lumped altogether. That's why different categories of farmers allow a much richer representation of reality, and since MAS specifically allows the interaction of heterogeneous agents, this gives much more richness to the model.

Mapping to Methodological Framework

Despite the importance of this feature in the real world, the aim is not to model and simulate individual real world farmers, but instances of types of farmers. This implies that first it is necessary to categorize the different kinds of farmers that are present in the system under study. In other words, the modeler must first identify and model a typology of farmers, in order to then populate the simulation with several instances of each type of farmer. To this aim, the proposed methodology includes the following steps:

- "*Identify Roles and Agent Types*": this step involves identifying the different roles the agents may play as well as the typology and the relations between roles and agent types. An important part of this step is to define the behavior of each agent type at each time step with UML Activity Diagrams.
- "*Domain Model Conceptualization*": this step builds a UML Class Diagram that not only helps in visually representing each agent type but also captures their structural aspects, like their attributes and relationships with other agent types, resources and environment.
- "*Define Agent Interaction*": this step captures the way in which the different agent types communicate with each other, including communication protocols and speech acts if necessary with UML Sequence Diagrams.

Feature: Organizations of Farmers

Along with the possibility to distinguish different kinds of farmers, it is often the case that farmers organize themselves. This implies a special kind of communication between those individuals that belong to the organization, differentiating them from those that do not.

Mapping to Methodological Framework

As a new agent type that is related to its members (which are other agent types). This involves a new concept in the "*Model Domain Conceptualization*" step with associations (aggregations) to its members in order to determine whether an agent instance is organized or not, and if it is, with whom. If belonging to an organization implies certain behavior in its members, then new roles can be defined for them in the "*Identify Roles and Agent Types*" step and agents should be able to change their behavior when playing this new role. Finally, if new forms of communication are introduced because of the organization, then it should be specified in the "*Define Agent Interaction*" step.

5.3 Capital Resources Related Features

These are goods and services created, purchased, or borrowed by the people associated with the farm to facilitate their exploitation of natural resources for agricultural production.

Feature: Goods and Services Consumed by Farmers

These are generally of two kinds: those which are used as inputs for production (like fertilizers and machinery), and those which are used as family consumption (like the electricity bill, health care or a new car). The former has to do with production resources while the latter with farmers' livelihood, needs and expectations.

Mapping to Methodological Framework

If any of these features are relevant concepts of the system as a whole, they should appear in the *"Domain Model Conceptualization"* step. Since they refer to consumption, which is simulated at each time step, they could also be modeled as part of each agent type behavior (therefore within UML Activity Diagrams of step "*Identify Roles and Agent Types*"). It is also generally the case that these goods and services have a cost that is not fixed for all time steps, so input parameters may also be needed (e.g. the cost of irrigation may be dependent on climate conditions modeled as input parameters), which are defined in the "*Simulation Configuration*" step.

Feature: Money, Savings, and Loans

Financial issues can take the form of savings, loans took by the farmers or farmers' organizations, the money that currently the farmer has on his/her pocket, the interest rate over a loan, etc. Variations on these elements are usually the result of production activities and family consumption, but also farmer's decisions. This feature is also commonly an important output parameter to monitor.

Mapping to Methodological Framework

The concepts involved in this feature may be divided into two: the activities that led to an increase/decrease (e.g. buying or selling) and the quantities (e.g. current amount of money the farmer has or how much debt he/she owes). The former can be modeled in the behavior of the farmer ("*Identify Roles and Agent Types*"), and the latter as an attribute in the *"Domain Model Conceptualization"* (e.g. an attribute 'current_money' in the concept 'Farmer'). If, on the other hand, it is relevant to keep track of the evolution of certain quantity and not only its effect over the current amount (e.g. to know each time the farmer took a loan) then it could appear as a separate concept (e.g. 'Loan') in the Domain Model. Regarding the need for monitoring this quantities (e.g. to know how the current balance of each farmer), the step "*Simulation Configuration*" allows for defining the output parameters that will be monitored at each time step.

5.4 Production Resources Related Features

These refer to the agricultural output of the farm, which becomes capital resources when sold, and residues (crops, manure) are nutrient inputs reinvested in the system. Even though the production resources may be diverse, when focusing on agro-ecosystems and farms the two that appear as central are crops and livestock.

Feature: Crops & Livestock

At the heart of any farm or productive unit there will be crops and/or livestock. These are what determine that the production process must take place over the land and not inside an industrial compound, making farmers live in a farm and not in a city. They are the center of the agro-ecosystem (together with farmers) since inputs are used by them, outputs are generated (including the money that results from commercializing them), behaviors and expectations largely affect them (through management practices) and their evolution over time may determine aspects like soil and water quality and natural processes.

Mapping to Methodological Framework

The mapping of these features is to model them as resources in the "*Model Environment and Resources*" step. They can either be associated to a certain land plot (as in the case of crops or farm animals), or not (livestock as a global not located quantity of animals). If a resource presents certain dynamics, these may be expressed in UML Activity Diagrams, much like the behavior of agents, but applied to resources. Since they are very relevant, they would also be (structurally) modeled in the "*Domain Model Conceptualization*". In order to initialize the simulation, the step "*Simulation Configuration*" would need to define the initial configuration of these resources (e.g. how many crops and crop types are present, and where), as well as define any relationship with input parameters, or if any monitoring is needed for them (output parameters).

5.5 Other Features

Also mentioned in **Section 2.2.1**, the following elements usually appear as a requirement when modeling agro-ecosystems.

Feature: Natural Processes

On the other end of the scale (compared to landscape) natural processes may be important in certain agro-ecosystem problems since they affect crops and livestock and are affected by weather and management practices. Examples of natural processes include organic matter evolution (how the organic matter of the soil evolve over time), and erosion (how much soil is lost due to rain or lack of cover plants). These processes are part of the biophysical subsystem.

Mapping to Methodological Framework

Since natural processes are not resources in themselves, but are closely related to them, it is useful to conceive both at the same time in the "*Model Environment and Resources*" step. Natural processes may be modeled by UML Activity Diagrams and be run at each time step, and the activities involved in this diagram will affect its related resources. As the "Crops & Livestock Feature" discussed before, resources may present certain dynamics to be run at each time step, so it should be clearly defined whether these dynamics actually represent natural processes or if they are only concerned with the resource itself. This implies that natural processes may be explicitly modeled as another feature of the agro-ecosystem, or implicitly considered when modeling resources. It may also be the case that the land itself presents certain dynamics of a natural process (e.g. organic matter evolution), and this could also be defined in the "*Model Environment and Resources*" step.

Feature: Landscape

Landscape represents the high level result of the interactions of all the previous features, and depending on the scale over which the modeler is to work, it can be of interest to have a certain view of it. Farmers can also be localized (geo-referenced) over the landscape, and patterns can be discovered as an emergent property of lower level interactions.

Mapping to Methodological Framework

The initial landscape can be determined by the modeler in the initial configuration, for example by determining how many plots are used for agriculture and how many for livestock. After the first time step occurs, leaving the initial configuration behind, the landscape will be the result of the interactions between the various components of the simulation, so rather than considering it an output of the simulation it should be considered as an outcome of it. This outcome should be easily viewable for example in a two-dimensional grid, where for each land cover a different color is assigned (using the corresponding visualization tool).

Feature: Market Prices and Evolution

Markets are always involved when commercializing goods and services. This could imply, for example, to know (and maybe keep track of) prices of crops and livestock.

Mapping to Methodological Framework

Since it is generally the case that the price over which production resources are sold is not controlled nor determined by the system under study, prices are considered as an external input to the simulation, determined in the "*Simulation Configuration*" step. In relation to this feature, although not considered as market price nor evolution, the cost of production of a farmer can be important, for example for determining farmer's profit (which is a commonly used output parameter), and can be defined as an attribute of each farmer in the "*Domain Model Conceptualization*" step.

Feature: Government Policies

Government decisions may greatly affect agro-ecosystems. Examples of this could be the increase of a certain tax over crop exports, or local authorities fostering new ways of farmers' organizations. The impact that government decisions/policies have over the system are generally so important that it is of interest to take them into account.

Mapping to Methodological Framework

This feature is not directly modeled into the simulation. This means there is no government agent type, because it is assumed that the government is outside the boundaries of the system under study. The interest will therefore be to compare the evolution of the simulation *with* and *without* the introduction of certain government policy. This requires modifying the simulation in order to take them into account. The modifications can range between changing input parameter values (e.g. because of an increase in taxes) and changing the behavior of agents (e.g. by introducing new ways of associations of farmers). These changes will then lead to the exploration of different scenarios (prospection).

5.6 Final Remarks

Table 7 summarizes the previous agro-ecosystem features and their corresponding representation under the methodological framework.

	Feature	Represented in step	As
Natural Resources	Land Plots	Model Domain Conceptualization	Environment concepts in the UML Class Diagram
	Water	A special kind of Land Plot	
	Climate	Simulation Configuration	Input parameters and Text Documents (if needed)
	Vegetation	Model Domain Conceptualization	Attributes of the Land Plot concepts or as environment concepts by themselves in the UML Class Diagram
Human Resources	Farmers' Types	Model Domain Conceptualization	Agent concepts in the UML Class Diagram
		Identify Roles & Agent Types	Text Document for role's identification and description and UML Activity Diagrams for behavior
		Define Agent Interaction	UML Sequence Diagrams for agent-agent interaction
	Organizations of Farmers	A special kind of Farmer Type	
Capital Resources	Goods & Services	Model Domain Conceptualization	Resource concepts in the UML Class Diagram
		Identify Roles & Agent Types	Actions in the UML Activity Diagram that correspond to those agent types that consume them.
	Money, Savings & Loans	The same as Goods & Services.	
Prod. Res.	Crops & Livestock	Model Domain Conceptualization	Resource concepts in the UML Class Diagram
		Model Environment & Resources	Resources
Other Features	Natural Processes	Model Environment & Resources	UML Activity Diagrams
	Landscape	The aggregation of all Land Plots	
	Market Prices & Evolution	Simulation Configuration	Input parameters and Text Documents (if needed)
	Government Policies	Involves changing the simulation models or input parameters and re-running them	

Table 7: Summary of agro-ecosystem features and their correspondence within the proposed methodological framework, including which step and which artifact.

This chapter presented the most common features that compose an agro-ecosystem and how they could be supported by the methodological framework. As a conclusion of this chapter, the methodological framework indeed gives support to all of these features.

The discussions in this chapter were given in a very general level, not giving any details on exactly how the steps and artifacts represent each feature. Therefore, the following chapter applies the methodological framework to a real-world case study.

6 Applying the Proposed Methodological Framework to a Case Study

The primary objective of this chapter is to have a first (but not full) validation of the methodological framework presented in this thesis by applying it to a relevant real-world problem.

Not necessarily all the artifacts are going to be needed to successfully develop this case study, since the methodological framework was not tailor-made to it, but to agro-ecosystems in general.

A secondary objective of this chapter is to show examples of real-world agro-ecosystems' problems and how they were modeled and represented by the different artifacts contained in the proposed methodological framework.

As stated in **Chapter 1**, the objective of this thesis is not to precisely define a complete and comprehensive software development methodology, but to define a methodological framework for agro-ecosystem simulations. Nevertheless, this chapter needs a specific order of steps to be followed to show how the methodological framework can be applied. This is why the steps presented in **Chapter 4** are here precisely sequenced, as shown in **Figure 10**. This doesn't mean that the steps should always be ordered this way.

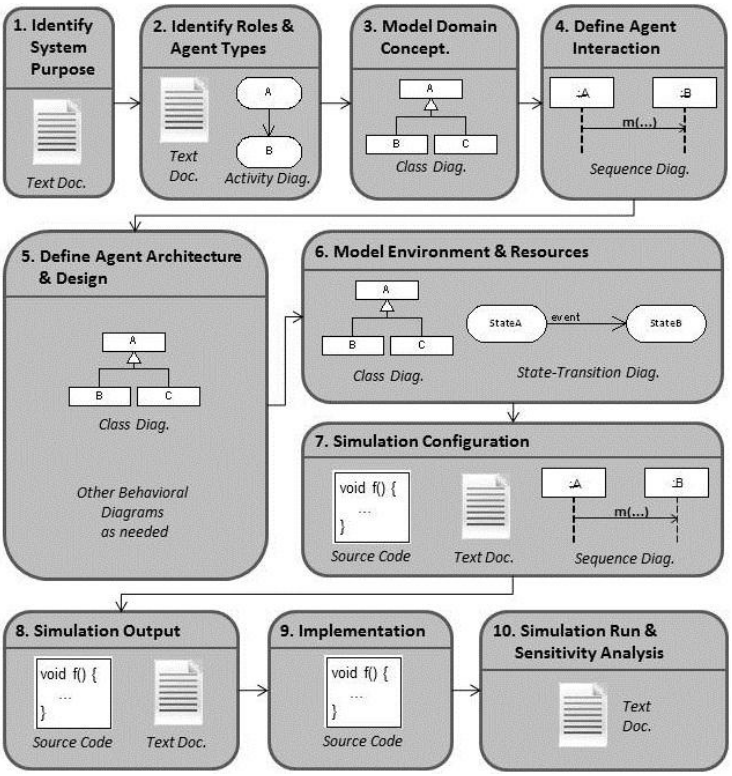


Figure 10: A possible order in which to follow the methodological framework. This order is followed for the case study.

This chapter starts with an introduction and motivation of the case study (Section 6.1), which is followed by the documentation of the steps of the proposed methodological framework (Sections 6.2 to 6.11), and ends with the most important conclusions of applying this methodology (Section 6.12).

This chapter assumes no previous knowledge in the field of agro-ecosystems, and particularly no knowledge of cattle-breeding farmers, which are the target population of the case study. Therefore descriptions or definitions could be given when considered necessary for understanding. Nevertheless, a Glossary is given in **Appendix C**.

6.1 Case Study Introduction

The case study presented in this chapter is a simplified version of a MAS model developed for a research project titled "Development, validation and evaluation of a modeling and simulation participatory methodology that contributes in the understanding and communication of the draught phenomena, and improves the adaptation capacity of livestock farmers in the basalt"[15].

This is a two-year project (2009-2010) financed by Instituto Nacional de Investigación Agropecuaria [INIA, 2010] and executed by Instituto Plan Agropecuario [IPA, 2010]. The former is an institution that develops and fosters agricultural research nation-wide and whose mission is "*to contribute to the development of farmers and the entire agricultural sector*" of Uruguay. The latter is an institution that mainly develops agricultural extension (to farmers) focused on livestock farmers and livestock production, and whose mission is "*to contribute to the sustainable and innovative development of livestock production and farmers, mainly small and medium size farmers, in order to improve their financial, family and human situations through training, extension, information and inter-institutional articulation*".

The research project was motivated due to severe draughts that affected the region (North Uruguay) in the last century, namely in 1916/17, 1942/43, 1964/65, 1988/89, 2005/06, among others. The severity of these jeopardizes farm sustainability in all of its three dimensions: economically (because of the loss of income and competitiveness), ecologically (because of cattle mortality and possible loss of grass variety), and socially (because of bankruptcy, emigration to cities and even suicides).

Also, the IPA team at North Uruguay knew that there was certain knowledge among farmers on how to adapt to these extreme situations. Moreover, some of this knowledge could be regarded as true adaptation strategies, but it was unclear about how they worked exactly, and which one was better.

This also evidenced the need for new methodological tools for the IPA team to work with, which also facilitate communication of these strategies (among farmers, but also between agronomists and farmers).

Finally, even though a draught is never good, the situation at North Uruguay makes it worse because of its basalt soils. This means that soils are very superficial (sometimes having only a few centimeters depth), and with basalt rock underneath them (therefore the name of the region). Because of this, the draught appears much more rapidly than in deeper soils, and its effect extends for much longer. **Figures 11 and 12** show examples of this effect over the soil and cattle.

Moreover, producers of this region cannot simply change to a different production activity, since the basalt soils are very poor and unproductive, making them not suitable for other agricultural productions (like soybean or rice).

The research project's general objective is "to contribute to the sustainability of basalt-soil livestock farmers" and its specific objective is "to evaluate a modeling and simulation participatory methodology for improving the understanding and communication of livestock farmer's adaptation strategies under draught phenomena".

This is one of the research projects in which the author of this thesis has worked as modeler.

[15] The project's description can be found here:
http://www.inia.org.uy/busqueda/proy_detalle.phtml?id=186&origen=1

Figure 11: Cattle trying to feed during the 2005/06 draught (picture taken by IPA).

Figure 12: A cow during the 2005/06 draught (picture taken by IPA).

The following sections develop this case study and correspond to those steps presented in **Figure 10**. They include the corresponding artifacts needed at each step.

6.2 Step 1: Identify System Purpose

The aim of this step is to define the purpose of the simulation, including its objective and questions to be answered. It contains just one artifact, a text document:

Document: System Purpose

System name:

The name of the simulation model to be built is *"Basalt Draught"*.

Context:

Several draughts have been suffered in Uruguay in the last century, especially affecting the Northern Uruguayan region, because of the presence of basalt-soils. These are very thin and are settled over basalt rock, so the draught phenomenon appears earlier, and it is suffered longer. This threatens livestock farmers' sustainability in all of its three dimensions: economically (because of the loss of competitiveness), ecologically (because of cattle mortality and possible loss of grass variety) and socially (because of migration to cities).

The IPA is very concerned about this to the point of proposing (and winning) an INIA-funded project that aims at applying a participatory modeling and simulation methodology for improving the understanding and communication of livestock farmer's adaptation strategies under draught phenomena.

Simulation objectives:

The objectives that were defined between the modelers and the IPA team were:

* To simulate the evolution of farmers under different draught strategies.
* To build prospective scenarios under the assumption that future conditions will be similar to previous (historical) ones.

Questions to be answered:

* Which draught strategy is more effective?
* For a certain draught strategy: What is the influence of the initial condition of producers (like the initial number of plots, cattle or initial budget)?
* Under what circumstances does a certain draught strategy succeed?
* What policies could be implemented in order to minimize the impact of a draught?
* Which could be the vulnerability indicators in order to build a vulnerability map of the basalt region?

Stakeholders:

IPA team: composed of six agronomists that work for IPA. Their role is to execute the project, trying to answer the above questions and reach the project's objectives.

Basalt livestock farmers: composed of around 20.000 small and medium-size producers (mainly family producers) that work and live at North Uruguay basalt region. These are considered breeders since they are the most common ones in the region.

INIA: the funding agency. Its role is to oversee the project's execution.

Government: although there is no one that represents the government, the results of the model could be of interest to policy-makers.

Target Population:

Basalt livestock farmers.

Modelers:

Pierre Bommel: member of the GREEN team at the CIRAD [CIRAD, 2010] with deep knowledge and expertise in the agent-based modeling and simulation domain. He holds a PhD in these topics and is nowadays based in Brazil, working as associate professor at the University of Brasilia. He also maintains the CORMAS simulation platform. His role is to be responsible for the modeling and simulation of the project.

Jorge Corral: assistant professor at the Engineering School of the University of the Republic, Uruguay. He holds an engineering diploma in computer science and for the last 3 years has been studying and working with agent-based models. His role is to assist Dr. Bommel and to use the project as a case study for his MSc thesis.

6.3 Step 2: Identify Roles and Agent Types

The aim of this step is to identify agent types and roles, especially agent behavior. The first artifact is a text document identifying roles and agent types:

Document: Roles and Agent Types

Roles:

Since the interest is in the modeling, simulation and prospective evolution of basalt farmers, it naturally follows that the first thing to do is to identify the different roles these farmers play. Even though it could have also been possible to include certain governmental role, this option was dismissed from the start, since the focus is placed on the farmers.

In order to identify the different farmer's roles, the IPA team had previously conducted several meetings with different producers in order to discuss the draught phenomenon and to share experiences and information. In all these meetings, the IPA team handed out polls that helped in the ordering and systematization of these experiences[16].

The result was that two very different strategies where identified that producers apply under draught conditions, so these were considered as the two roles identified: Proactive Producers and Reactive Producers.

Conceptually (without going into details), the Proactive Producer looks mainly at grass availability in order to trigger draught-related activities, and gives priority to cattle condition over cattle quantity. This means that they would rather have less animals but well fed ones, than more animals with less food (grass). The Reactive Producer looks mainly at the cattle condition in order to trigger draught-related activities, and gives priority to the number of animals in its plots.

Draught-related activities include supplementing the animals (e.g. with grains), rent more plots in order to have more grass availability (more food), make unusual sells (this means to sell animals that under normal circumstances would not be sold), among others.

Because the activities that take place within the context of a draught may greatly vary among the different seasons, the time step of the simulation must be set to one season.

Agent Types:

Because the model to be built was not going to consider the possibility to change the draught strategy of any producer, and because each producer only follows one strategy, it was not considered necessary to differentiate between Roles and Agent Types. Therefore, the agent types (agent classes) that are to be considered coincide with the identified roles. This means that one agent type is called Proactive while the other agent type is called Reactive.

Interestingly, this case study does not require two-dimensional grid visualization like many other MAS simulations do. This is because there is no need to work with coordinates (locations) since, for instance, it does not matter where the cattle is, and the entire plot is assumed to have an homogeneous distribution of animals (assuming they will walk around and uniformly eat the grass). Also because the simulation runs with one producer at a time, there is no need to geographically locate them; actually it is not important where the farm is located, as long as it is in the basalt region.

The second artifact is a UML Activity Diagram for each agent type. In this case, since agents strategies vary among the different seasons, there will be four diagrams for each of the two strategies. For clarity reasons only two diagrams are shown that best represent the differences between both strategies: the activities performed in winter by each agent type. Before presenting the diagrams, a brief introduction is given.

Even though the draught strategies of the two types of producers are different, they can both do the same activities. The following is a list of activities that both agent types may perform at winter (since both diagrams will refer to that season):

- **Pay farm costs**: the first thing a producer does when a season starts is to pay over the costs of the previous season's activities. These costs only include those associated to the exploitation of the farm, not including family costs. The effect of this is to decrease the producer's balance.

[16] More than a thousand producers attended these meetings during the 2005/2006 and 2008/2009 draughts.

- **Pay interests**: only if the balance is negative the producer pays a certain percentage of its debt (financial cost). This is relevant information since the balance is one important outcome of the simulation and because producers never go in bankruptcy (it could easily be added a rule that makes a producer go bankrupt if his balance is negative). As in the previous activity, the effect is a decrease in the balance.
- **Mating**: since both types of producers are considered breeders, they need to mate the cows (both empty as well as lactating) with a bull in order to make them pregnant. The effect of this activity is to mark up all mated cows in order to know afterwards if they got pregnant or not. This will depend on the cow's condition score (healthier cows are more likely to get pregnant than weaker ones).
- **Sell steers**: because producers are considered to be breeders, they do not make much effort on retaining steers (young bulls). It is worth noting that these steers come from the previous seasonal step, and not from the mating explained above. The effect of this activity is to decrease the number of steers for that producer and to increase its balance because of the money earned by the sale.
- **Sell empty cows**: winter is the worst season facing a draught, since it is when the grass grows more slowly (because of the decrease in sun radiation). This justifies selling some cows for two reasons: first for lowering the total number of animals to face the winter, and second to increase producer's income. Empty cows are preferred since pregnant cows have more value and lactating ones are needed by their calves.
- **Adjust stocking rate**: analogous to the previous activity, adjusting stocking rate has the effect of decreasing the total number of animals and increasing income. Nevertheless, both agent types will choose different rates: while one will focus on retaining animals (reactive) the other one will focus on maintaining condition score (proactive) so it will reduce more than the former. This reduction only takes place in winter because of the reasons explained above.
- **Graze**: when the effects of a draught start to appear, grazing is a natural option for producers. It implies moving a certain amount of animals to another (rented) plot in order to increase grass availability. The effect of this activity is that animals are best fed but the producer must pay for the rented plot. Here again, each agent type is willing to pay more or less for each rented plot for grazing.
- **Return from graze**: after a draught or if the grazing prices are too high, producers may choose to return their animals from rented graze back into their own farms. The effect is to reduce the total available area (and grass availability) for animals to feed, and stop paying the corresponding grazing prices.
- **Extraordinary sale**: when the effect of a draught is severe, producers may do extraordinary sales that under other circumstances they would not. Nevertheless, the prices paid for the animals must be considered "good". In the simulation a price for a certain category of animal is considered to be "good" if it is similar to those prices paid over the last two years. The effect of this activity is to reduce the animal stock and to increase producer's balance.
- **Supplement**: if prices are not good enough for an extraordinary sale, producers may supplement their animals. This generally means to buy special food (including grains) with very high nutritive value. The effect of this activity is to increase the condition score of the animals and to decrease producer's balance (because he has to pay for it). Here again, the circumstances under which both agent types choose to supplement their cattle will be different: while one will focus on animal survival (reactive) the other one will focus on animal maintenance (proactive).

It is important to note that even though both agent types *can* do all of these activities, this does not mean that they will both choose the same activities at the same time, since often their criteria for deciding what to do will differ. This translates, for example, into differences in the guards of the decisions in the UML Activity Diagram, or even not including an activity, like mating in winter, which is not a choice for the proactive agent type (he only mates his cattle in summer).

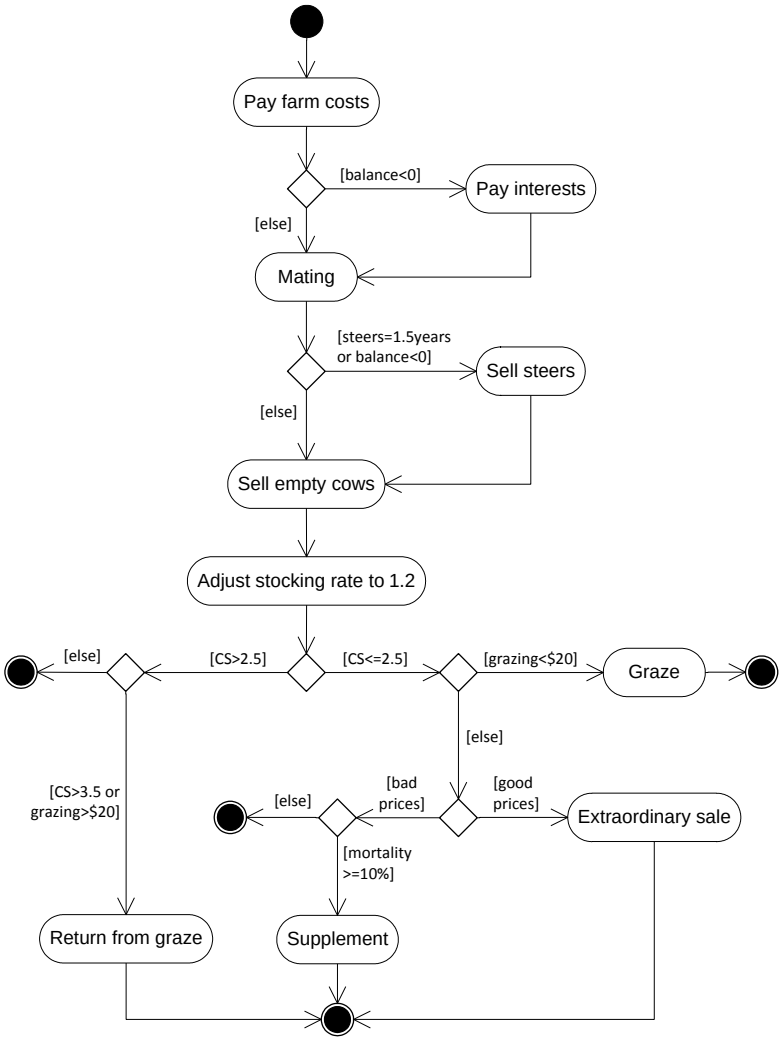


Figure 13: Winter strategy for a reactive producer (CS stands for Condition Score).

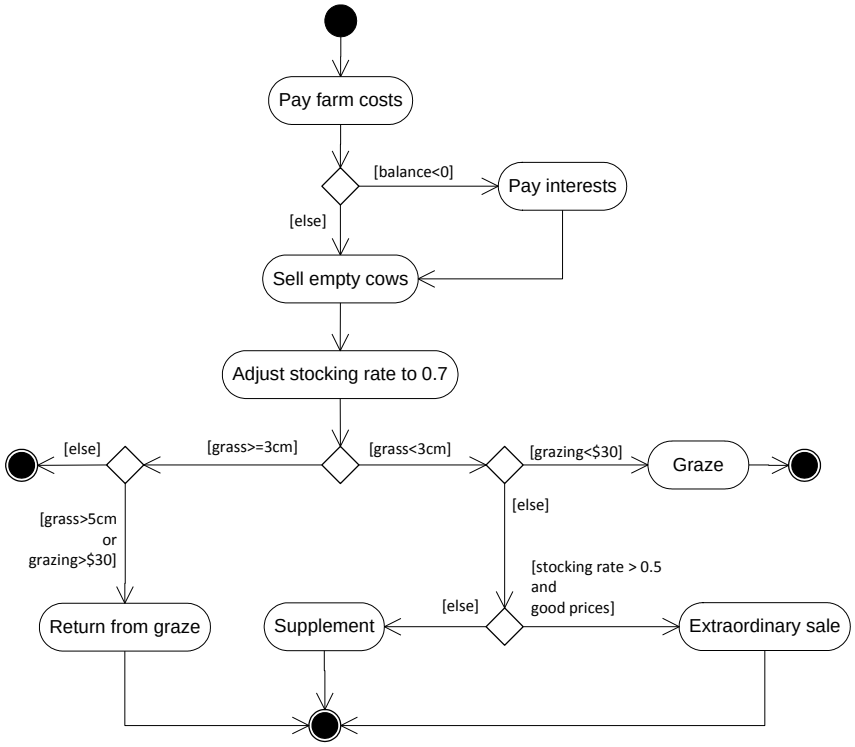


Figure 14: Winter strategy for a proactive producer.

6.4 Step 3: Model Domain Conceptualization

The aim of this step is to depict the structure of the problem, including entities and relationships. A simplified[17] version of the domain model developed is shown in **Figure 15**:

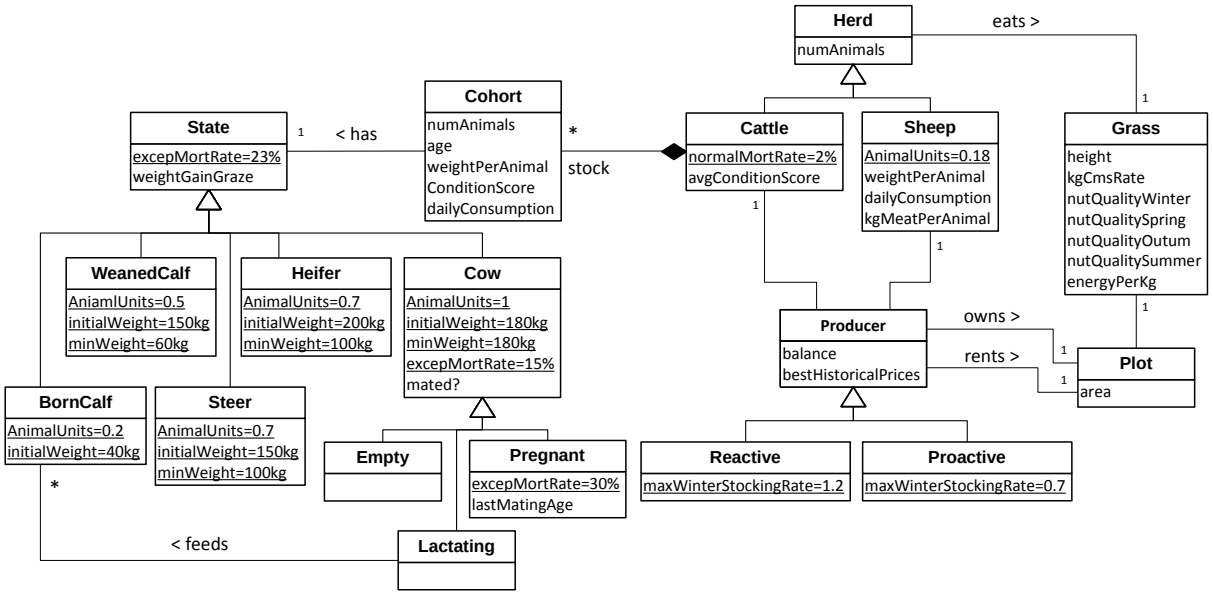


Figure 15: Domain Model for the case study.

The following list briefly introduces each concept, organizing them as agents, resources and environment:

Agent-related concepts:

- **Producer**: the general concept for both agent types. Its main attribute is the current balance of each agent instance.
- **Reactive & Proactive**: they both represent the two different agent types identified in Step 2. The figure shows how they differ in the maximum winter stocking rate (one adjusts to 1.2 at the beginning of winter while the other to 0.7). Although not included in the Domain Model (but in the Class Diagram of Step 5) they will both include the different steps for each season (as methods of their corresponding classes).

Resources-related concepts:

- **Herd, Cattle & Sheep**: even though the farmers represented in the simulation were described as cattle breeders, they generally also have a flock of sheep. This is especially useful in draught scenarios since sheep are nearly not affected by draughts since they are able to get more grass than cattle (because of their teeth). Nevertheless, the focus is on cattle, and this is why several categories were defined, since the average condition score of the cattle herd is important to make decisions, while sheep are considered to be all the same. This means that the flock of sheep never evolves over time, it is always fixed (this assumption was validated with the farmers and was found to be more suitable than it looks). Finally, it is worth noting that each producer has only one herd of cattle and one herd (flock) of sheep.
- **Cohort & State**: as already discussed, a strong focus is placed in the cattle herd (because producers are breeders). This implies a fine-grained modeling of cattle evolution that may have led to represent individual animals in the model. Nevertheless it was not considered necessary (or useful) to individually identify each one of the animals a producer has (e.g. by ear tags), neither to consider the entire cattle herd as one uniform set of animals (like the sheep herd) because this would lumped altogether different categories of bovines, with different age, weight, sex, condition score, etc. Therefore the solution was in the middle: lump together animals (bovines) that have the same category, age, weight, etc. This is represented

[17] The simplification only involved attributes, not the structure of the Domain Model.

by the Cohort concept (which means "a group of people who share a characteristic, usually age"). The characteristics shared by the cohort are represented by the State concept, further discussed below. An example of a cohort instance is a set of 10 newly born calves, all siblings, sharing the same category (BornCalf) and the same age (e.g. 1 month).

- **Cow, Empty, Pregnant & Lactating**: these concepts represent the different states in which a cow can be, therefore in which a cohort of cows can be. Empty means the cow is not Pregnant and Lactating means the cow is feeding its newly born calves (which are known to her thanks to the "feeds" association). At each one of these states the dynamics of the simulation are different. The fertility rate that is obtained after mating (the percentage of cows of a cohort that actually get pregnant) as well as the exceptional mortality rate (the percentage of cows of a cohort that die because of starvation) are calculated by formulas that depend not only on the cow's condition score, but also on their state (if they are lactating or empty). The energy that a cow needs also depends on these categories: the amount of grass consumed (therefore the energy obtained) is not the same for a lactating cow than for an empty one. Finally, the prices over which the bovines are sold also depend on their categories. All of these justify the fined-grained representation of cows and their categories in the model.
- **Borned Calf, Weaned Calf, Steer & Heifer**: similarly, it is important to differentiate when a born calf is still with its mother than when it is weaned (taken away from her and given special supplement) in order to let her mate again with a bull. Also the distinction between steers (males) and heifers (females) is important because producers are mainly breeders, and they tend to sell their steers but keep their heifers, since the latter will turn into cows.

Environment-related concepts:

- **Plot**: each farmer owns one 500 ha plot of land over which he puts his animals (represented by the "owns" association). Nevertheless, the Graze activity allows him to rent other plots (of variable size) when conditions are harsh (represented by the "rents" association). An assumption was made that plots have no divisions (fields) and that the availability and quality of the grass is the same on all plots (owned and rented). In other terms, the farm is not spatialized. This choice comes from the fact that the focus of the model is not on individual cows but rather at the level of the herd.
- **Grass**: represents the (only) kind of grass that is available to all animals and all producers. The grass grows depending mainly on three factors: the season (e.g. it grows much more in summer than in winter), the weather on that season (e.g. the more rain the better), and the previous availability of grass. The importance of the seasons in these calculations helped in determining the seasonal time step of the entire simulation. The rates over which the grass grows for each time step is calculated (seasonally) considering the weather of that season (an external factor that is taken into the simulation via input parameters) and the grass availability at the end of the previous step (an internal factor that is affected by animal consumption). Here it is also very important to have a fine-grained representation of reality since grass availability is what ultimately affects animals' condition score, and the grass-animal dynamic was needed to be very fine-tuned and confident before introducing producer's decisions (**Section 6.10** describes the phases in which the simulation was developed).

6.5 Step 4: Define Agent Interaction

The aim of this step is to determine when, how and what the different agents will communicate with each other.

Because the aim of the project does not involve nor require interaction between the different producers, this step does not apply for this case study.

6.6 Step 5: Define Agent Architecture and Design

The aim of this step is to define agent design (structure and behavior) in order to fulfill its perceive/reason/act cycle, within a simulation framework software package.

As introduced in **Section 2.3.2**, agent architectures can be classified in an abstract way (closer to reality) as deliberative, reactive or hybrids; which are further supported by concrete software architectures (closer to implementation). This case study follows a reactive architecture, so there will be no explicit representations of beliefs, desires, intentions or environment, but rather each agent will act upon a set of predefined rules, which were already shown in the Activity Diagrams of **Step 2** (e.g. if it is winter, the grass is 2cms and grazing is cheap, then put the animals to graze).

Regarding the concrete architecture referred to in **Section 2.3.2**, which implements the abstract architecture in a certain software environment, and since this case study uses CORMAS, then an analysis should be done in order to determine which CORMAS classes should be used or extended so that it supports the reactive architecture. In other words, design each agent type by means of new classes, extended CORMAS classes and/or interfaces.

Before presenting the UML Class Diagram showing the final agent design, and which CORMAS classes were extended, a brief introduction to those CORMAS classes will be made. **Figure 16** shows an excerpt of CORMAS main classes and their relationships (attributes, operations and auxiliary classes where omitted).

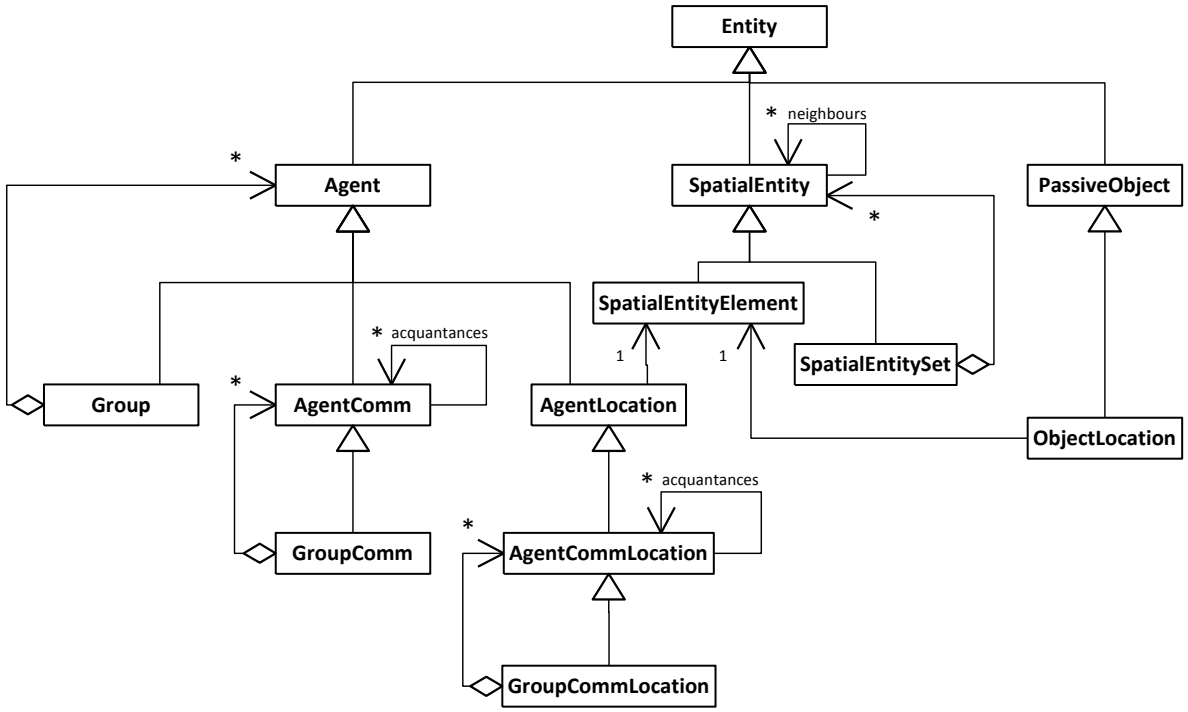


Figure 16: UML Class Diagram for CORMAS main classes.

As the figure shows, CORMAS natively supports the three main MAS concepts discussed in **Section 4.2**: agents (the **Agent** class and subclasses), environment (the **SpatialEntity** class) and resources (the **PassiveObject** class and its subclass).

Furthermore, CORMAS defines three dimensions that can be applied in combination to these three concepts: the ability to group together, communicate or locate. The following table summarizes these three dimensions and shows the relation between these and certain CORMAS entities:

CORMAS class name	Description	Represents an entity that is capable of grouping with others?	Represents an entity that is spatially located?	Represents an entity that is capable of communicating with others?
Entity	General base class	N/A	N/A	N/A
Agent	Agent base class	N/A	N/A	N/A
SpatialEntity	Spatial entities' base class	✗	✓	✗
SpatialEntityElement	A spatial entity	✗	✓	✗
SpatialEntitySet	Group of spatial entities	✓	✓	✗
PassiveObject	Other entities' base class	✗	✗	✗
Group	Group of agents	✓	✗	✗
AgentComm	A communicating agent	✗	✗	✓
AgentLocation	A located agent	✗	✓	✗
ObjectLocation	A located passive entity	✗	✓	✗
GroupComm	Group of communicating agents	✓	✗	✓
AgentCommLocation	A located and communicating agent	✗	✓	✓
GroupCommLocation	Group of located and communicating agents	✓	✓	✓

Table 8: CORMAS main classes description and relation to the three already-defined dimensions: group, location and communication.

Since producers are only located at their farm and need no communication with other producers, nor they need to group, both agent classes extend the CORMAS **AgentLocation** class indirectly, by making the **Producer** class extend it. This is shown in **Figure 17**:

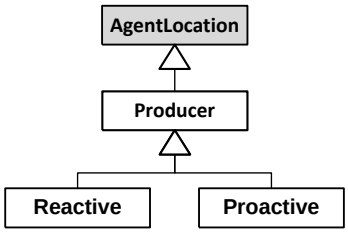


Figure 17: Both agent types extend the CORMAS **AgentLocation** class.

The following step includes all the other classes included in this case study and completes the entire UML Class Diagram for the case study.

6.7 Step 6: Model Environment and Resources

The aim of this step is to determine behavioral aspects (evolution) of resources and environment, and to complete structural aspects.

Regarding the structural design of the environment-related classes, **Figure 18** shows the result, which simply implies extending CORMAS **SpatialEntityElement** class.

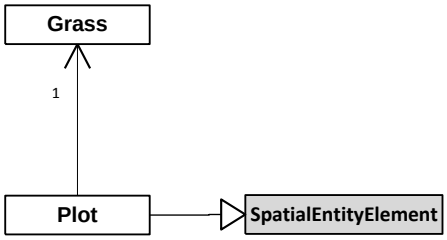


Figure 18: The Plot class extends the CORMAS **SpatialEntityElement** class.

The dynamics of grass growth required a thorough analysis, which involved obtaining real data about grass growth rates for each station (obtained by satellite images) and making it dependent on the initial height for each season (for example, it would not be the same growth if the initial height of grass in winter was 1cm or 5cms). This analysis is out of the scope of this thesis since it involved complex but purely agronomical and biophysical calculations (and mathematics), but no software complexity.

It is worth noting at this point that other environment-related features like the weather where considered as input to the simulation, so the only impact on the design was to include a class called Environment (shown in **Figure 21**) with methods that, for each time step, returned the weather conditions for that step. As it was discussed in **Section 5.1**, climate is considered to be outside the system, so its influence is in the form of time-series input parameters, so it will be discussed within the *Simulation Configuration* step.

Concerning the behavior of resources, in this case, animals, the main challenge was first to understand, model and implement their natural lifecycle, like feeding, mating and reproduction, and then to include farmers' decisions into them, like sales. **Figure 19** shows a UML State-Transition Diagram (a kind of UML Behavioral Diagram) containing cattle lifecycle for the Reactive farmer. Technically, the diagram shows the different states for any object that is instance of the **State** class, which in reality means the different states of a cohort.

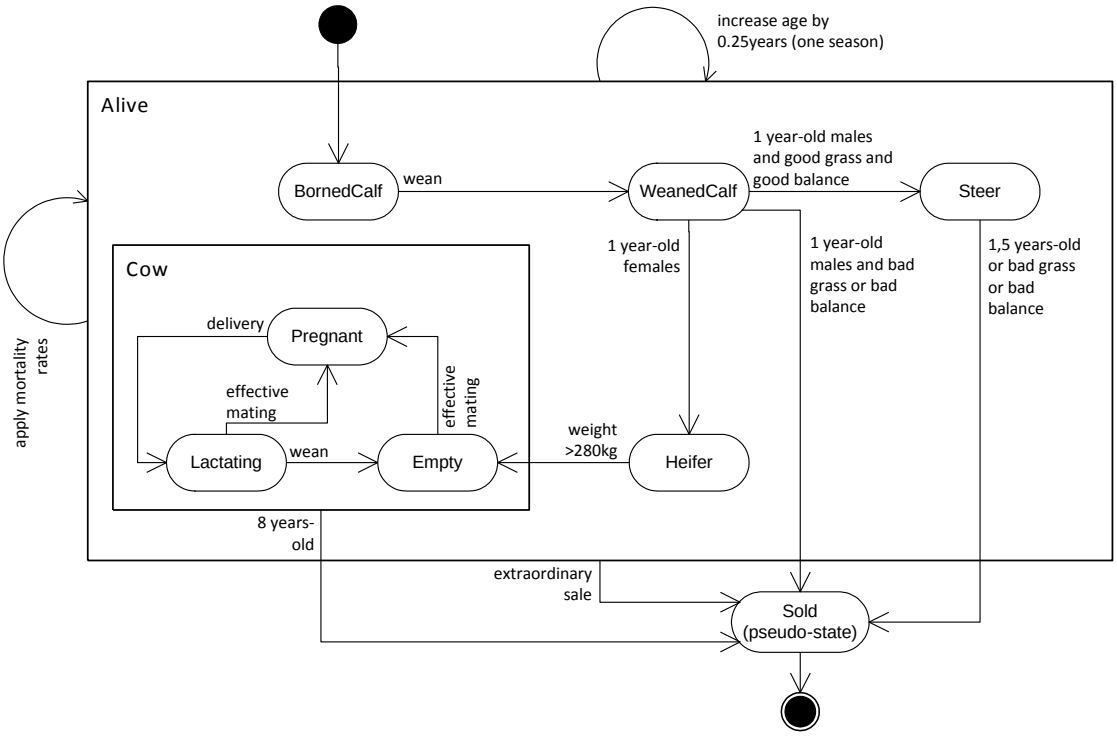


Figure 19: UML State-Transition Diagram showing cattle lifecycle for a Reactive Producer.

Finally, **Figure 20** shows all the classes (except from the already mentioned **Environment** class that will be discussed below) that compose the design of the case study, including the extended CORMAS classes (all attributes and operations are omitted).

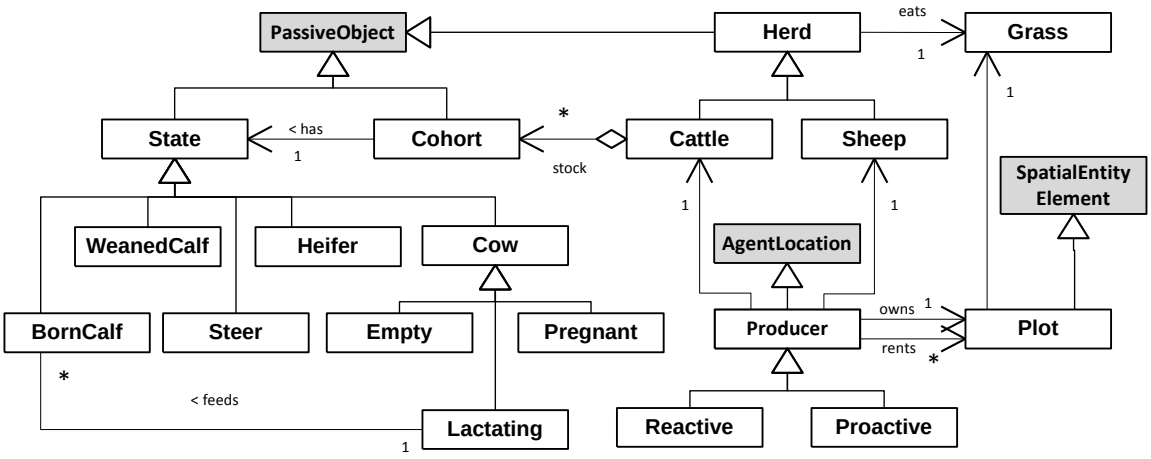


Figure 20: UML Class Diagram for the case study showing extended CORMAS classes (shown in color) and omitting attributes and operations.

The Environment class shown in **Figure 21**, represents external features such as the weather, cattle and sheep prices, grass growth rates, etc. for each season. Therefore, each time a class needs any of this information for the current season, it asks for it to the **Environment** class.

This class is also strongly related to the input parameters, since the information it contains represents external features. This issue is further discussed in the following step.

Environment
-seasons : Vector -currentSeason : String -years : Vector -currentYear : Integer -cattlePrices : Matrix -grassGrowth : Vector -grazeAvailability : Vector -grazePrices : Vector -weatherCoeff : Vector -sheepWoolPrice : Double -sheepMeatPrice : Double -supplementPrices : Vector
+initSeasons() +initYears() +initCattlePrices() +initGrassGrowth() +initGrazeAvailability() +initGrazePrices() +initWeatherCoeff() +getSeason() : String +getYear() : Integer +getCattlePrice(in State) : Double +getGrassGrowth() : Double +isGrazeAvailable() : Boolean +getGrazePrice() : Double +getWeatherCoeff() : Double +getSheepWoolPrice() : Double +getSheepMeatPrice() : Double +getSupplementPrice() : Double

Figure 21: UML Class Diagram showing (part of) the Environment class that designs how the input parameters are used within the simulation. The initXXX() methods correspond to the initialization of the input parameters and is discussed in the following step.

6.8 Step 7: Simulation Configuration

The aim of this step is to define those fundamental elements that will enable a simulation to be run. These are (as **Section 4.4** introduced):

- **Initial configuration**, which is explained in the following text document but is mainly represented in raw source code;
- **Time step definition**, which is defined in the text document;
- **Task scheduling**, which is represented by a UML Sequence Diagram, and
- **Input parameters**, which like the initial configuration, is explained in the text document but represented directly in source code.

Document: Simulation Configuration

Initial Configuration:

The initial configuration for this case study basically involves defining how many cows of each kind will be at time zero (how many cohorts), how many sheep will be (the number of sheep of the flock is fixed during the simulation), the initial grass height, the initial budget for the producer (how much money he has in his pocket at time zero), and initial input parameters (which are discussed below), which will depend on the initial season selected.

An important issue is to be able to easily modify this initial configuration in order to test different scenarios, each of them varying its initial state. To this end, CORMAS provides a simple definition of different "init" methods that can be written (coded) altogether and then simply select one each time the simulation is run. This enables, for example, to inspect if different trajectories emerge for a certain type of producer by only changing his initial state.

Time Step Definition:

Like the **Roles and Agent Types Document** introduced, the time step for the case study was defined to be a season (i.e. 3 months) because the activities that could take place within a draught may greatly vary from season to season. This means that four steps actually represent the passing of one year.

Task Scheduling:

At each time step (season) the following events occur (it should be noted that these events are assumed to be executed the first day of the season since it is a simulation and not the real world):

1) The weather and prices are updated (refer to Input Parameters below) for the current season, meaning that this new weather and prices will be valid through this season.

2) The producer decides and performs its seasonal activities. This implies choosing the UML Activity Diagram that corresponds to the producer's type and current season. This is also called the Producer's *Step Method* in CORMAS. Here is worth noting that the producer decides and immediately executes those decisions (again, for the sake of the simulation).

3) The grass grows for this season. Here, even though the grass grows immediately, the one consumed by the animals (in the following two events) is calculated with the average grass height in order to make it more real; otherwise the animals will always eat at maximum grass height.

4) The sheep evolve. This only means that they feed themselves (eat grass) and that they produce certain amount of wool and meat (refer to the Domain Model Conceptualization Step for the attributes of the sheep flock).

5) The cattle evolve. This means not only they feed but also reproduce, since cattle lifecycle is relevant for the case study (as already explained).

See **Figure 22** for a UML Sequence Diagram showing these events. The Scheduler is actually a CORMAS class that orchestrates all seasonal activities, invoking the necessary methods over their corresponding classes.

Input Parameters:

The input parameters needed for this case study already appeared in the **Environment** class shown in the previous step, namely: the **years** during which the simulation will be run (this

information is not usually needed in MAS simulations but in this case, historical real-world data was used as time-series input parameters, so it was important to map that data within the year and season in which it occurred in order to find any inconsistencies); the **weather**, taken from historical real-world data (this was represented as a coefficient for each season that multiplied by the grass growth for that season determines the final grass growth); the **prices** for cattle and sheep (for sheep it is maintained always the same, but for cattle it varies depending on the season using the historical real-world data that was gathered, and on the category of the animal since it is not the same the price for a newly born calf than for a fully grown cow); the **grazing** availability and its rent prices (that is, the price that producers have to pay per animal, per season, in order to put them into grazing); and the **supplement** prices (how much it costs to feed an animal during a season with supplement).

All of these data was initialized (in their corresponding **initXXX()** method) using historical real-world data that was gathered and/or estimated for the basalt region, making the simulation more accurate for this region.

Finally, it is worth noting that a draught condition implies that the grass growth is low (because of poor rain), the weather coefficient is also low (therefore grass height will be compromised), grazing prices are high (because the demand for grazing rises, pushing its prices), and cattle prices for all categories are low (because cattle supply increases, making prices decrease). Such a condition can then be easily forced into the simulation, simply by changing these input parameter values before running it.

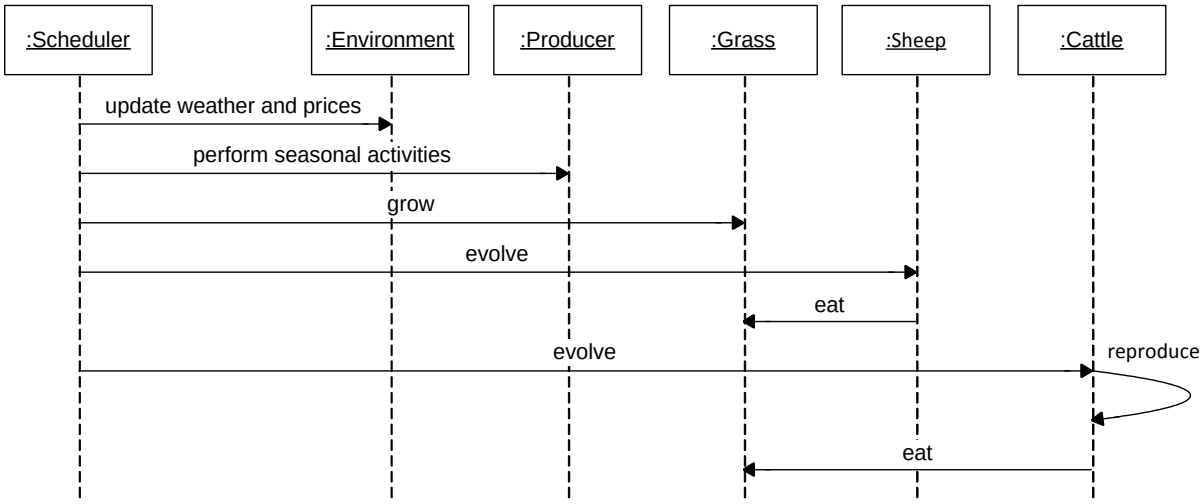


Figure 22: UML Sequence Diagram showing the task scheduling for each season.

6.9 Step 8: Simulation Output

The aim of this step is to define the output that the simulation will produce along with its visual representation. **Table 9** summarizes the output parameters that where defined for the case study, along with the desired visual representation for each output parameter (all of them are shown on a per-season basis):

Output	Description	Visualization
Cattle-generated Income	Shows the income generated by selling cattle in this season. This allows for inspecting which of the two kinds of animals (sheep and cattle) is giving more profit to the producer.	Graph
Sheep-generated Income	Analogous to the previous, but for sheep.	Graph
Producer's Income	This value includes the previous two (with positive sign) and exploitation costs (with negative sign), so it can be positive (the producer gained money in the season), or negative (he lost money).	Graph
Producer's Balance	Show how much money the producer has in his pocket (previous balance + cattle-generated income + sheep-generated income – exploitation costs).	Graph
Producer's Total Assets	This value represents how much the producer is worth (value of cattle herd + value of sheep herd + current balance).	Graph
Avg. Condition Score (only for cattle)	Shows the average condition score of the entire cattle herd. This is relevant to get a picture of the animal's overall health. It is not necessary for sheep since they are assumed not to be affected by draught (so they maintain their condition score, so it is not interesting to keep track of a constant-value).	Graph
Cattle Stock	How many animals of each category (state) the producer has.	Graph
Cattle Death	How much cattle is dying because of draught phenomena, (the natural mortality rate should be excluded from this figure).	Graph
Weather	Show the current weather conditions in order to properly interpret and analyze the output.	Graph
Season & Year	For the same reason mentioned before, show the current season and year (e.g.: Fall-2008).	Label

Table 9: Output parameters defined for the case study along with their visualization requirements.

The previous output parameters where chosen because in combination, they can help answering the case study questions (and therefore their objective) defined in **Step 1**.

6.10 Step 9: Implementation

The aim of this step is to codify the simulation in a programming language. The implementation phase of this case study involved three iterations, each achieving a fully functional, but reduced version:

- **Grass Model**: the first version of the model consisted only of the grass evolution over time. This decision was supported on the grounds that grass growth was very important in the model since it determined how much food the animals will have at each season, and the calculations needed for this were not trivial. The results that this first version yielded were the grass height at each season, according to the weather conditions present in that season. After validating these results with the experts, the team defined the scope of the second version: to include cattle and sheep but not yet producers.
- **Wild Model**: this second version of the model added the animals to the first one, but as if they lived wildly in the farm, that is, without producers managing them[18]. This let the team test and validate with the experts the behavior of the animals regarding how much they were eating (and weighting), how the reproduction cycle was done, etc. The only thing that was different from a truly wild model was that mating occurred only once per year and not permanently as it would be the case in a real wild condition. After getting confidence on the grass-animal interaction, the team moved on to the third and final version: include the producers.
- **Final Model**: the last version of the model added human actions over the previous wild model version. Specifically, each agent-type (Proactive and Reactive) was now able to behave as their role specified (Step 2) following the activities defined in its UML Activity Diagram. Producer's actions directly affect the animals (like mating, supplement, sales) and indirectly affect grass (through animals).

[18] Actually, within the team, this second version was called the "*Hernandarias Version*", after the man who introduced cattle in Uruguay back in the 17th century.

This iterative and incremental approach helped to divide the workload (and not to manage all features together from zero), and to have early feedback from the experts (since there were two intermediate versions that could be validated long before the final one).

As introduced before, this case study was developed using the CORMAS simulation platform, which is a MAS software package that allows developing this kind of simulations. The language in which CORMAS was developed was Smalltak [SMALLTALK, 2010] and so was this case study. The development environment and compiler were Cincom's VisualWorks [CINCOM, 2010].

Regarding the programming that was required to make the final version complete and running, and since VisualWorks does not have a place where the programmer can look at the entire source code all at once (it always shows a partial view of the code), it is not easy to estimate how many lines of code the final version has. However, VisualWorks converts all code into a single XML document for saving and restoring the project, so the lines of XML code could be fairly comparable to the lines of actual Smalltalk code. The XML document for this case study has 6.000 lines.

Another measure could be made by considering the number of classes included in the final version, which are 19, as Step 6 showed before (including the **Environment** class).

The implementation of the case study took approximately 7 weeks (considering working days) of the entire team (including the two modelers), with a full-time dedication, divided in:

- **From March 11th to March 13th of 2009**: project definition and first discussions in Tandil, Argentina.
- **From October 5th to October 9th of 2009**: Formal project kick-off. Involved several discussions and the main output were UML diagrams, followed by small prototype versions in CORMAS as proof of concept (Salto, Uruguay).
- **From November 30th to December 5th of 2009**: Grass Model construction (1st version) and validation with stakeholders (Salto, Uruguay).
- **From February 25th to March 5th of 2010**: Wild Model construction (2nd version) and validation with stakeholders (Salto and Artigas, Uruguay).
- **From September 11th to September 17th of 2010**: Final Model construction (3rd version) and validation workshop with stakeholders (mainly with producers in Salto and Artigas, Uruguay).
- **To come: From March 14th to March 18th of 2011**: Debugging of Final Model and construction of Interactive Model for use in a workshop with producers in Salto, Uruguay (the concept of Interactive Model is introduced in the next step).

6.11 Step 10: Simulation Run and Sensitivity Analysis

The aim of this step is to achieve the simulation's objective and answer its questions. As it was shown in the previous step, the Final Model Version is yet to be further debugged in order to get full confidence in its results, and to use it as a tool for the March 2011 workshop with producers. In this workshop, producers will use the simulation as a game where they are represented by their avatar (Proactive or Reactive) inside the model.

For this workshop, a more interactive version of the model is being constructed, which can be called Interactive Model, basically consisting of the same as the Final Model Version but letting the user (in this case the producers) pause and interact with the simulation. The only difference between the Interactive Model and the Final Model Version is that while the latter automatically runs through the seasons, the former pauses at each season and asks the user what to do (from a pool of valid options). This means that instead of automatically running without interaction through a whole period (say 20 years) the Interactive Model asks for user decision making. This was thought as a way to engage the producers more actively in this "game", while opening the black-box that the Final Model Version may seem to them at the same time (in other words, passing from a black-box to a white-box).

6.12 Conclusions of Applying the Methodological Framework to the Case Study

This section provides some conclusions and final remarks about applying the methodological framework to this case study.

Overall, the methodological framework successfully supported the development of the case study. Knowing what steps should be taken and what artifacts should be constructed provides confidence, saves time, and keeps people focused. Having a mapping between the most common features found in an agro-ecosystem and how they can be represented using the methodological framework was shown to be of high value.

The iterative and incremental approach was not initially considered in the development of the case study, but once the scope and complexity were more clearly identified, this approach was taken and proved to give not only a macro-level organization (which features should be considered in which version) but also much more confidence on each version before getting into the other. This confidence was the result of receiving early feedback and of having the previous version as the tested and trusted baseline over which to build the next one.

In a truly interdisciplinary team, like the one that developed this case study, to share a basic vocabulary and to have a common understanding of the basic concepts is crucial. To this end, the Domain Model served as a glossary of the most important concepts and provided this common understanding much in the same way an ontology would do.

Another interesting conclusion was reached in the use of UML diagrams. At the beginning of the project, all IPA members were convinced that they all shared the same knowledge about basalt breeders. However, the use of UML, particularly the Activity Diagrams in this case, demanded experts to explicit their knowledge and assumptions in order to unambiguously build the diagrams. From this process, several contradictory opinions emerged between the different experts, so several discussions took place to arrive at the final versions of these diagrams. This is an example where implicit and explicit/shared knowledge did not coincide, and the use of a formalism showed this situation.

Regarding the producers, along the several workshops, they felt they were being accurately represented by the UML Activity Diagrams, and even when other variants were proposed (e.g. like considering a third kind of producer as the result of a mix of the other two) the great majority did coincide that the two types of agents (Reactive and Proactive) indeed represented them.

Other aspects of applying the methodological framework did not go so smoothly, and are revisited in **Chapter 7** as future work.

First, the architecture chosen for the case study, as **Step 5** showed, was a reactive agent-architecture. This is by far the simplest architecture of all three (reactive, deliberative and hybrid). Nevertheless, the system as a whole already became quite complex. This means that if another architectural style was chosen, the final solution would have been even more complex. However, it would be of great value to try using the other two along with the methodological framework.

Second, the dependency between steps is not as straightforward as the methodological frameworks suggests. Particularly the time step definition, for example, could be defined earlier. These aspects should be revised in the context of a more comprehensive software development methodology.

Third, input parameters proved to be very important in the development. However, they got little attention in the methodological framework. Even though it does not seem easy to further propose structures, or rules for them because of their very disparate nature, a more thorough analysis should be done. This includes their definition (which input parameters the system will need?), their representation (which data structures and functions they will have?), initialization and evolution over time.

Finally, because of the very nature of the case study, **Step 4** (*Define Agent Interaction*) could not be applied; it was not needed in this case. This left behind the opportunity to test how the methodological framework supports the definition of the interaction between multiple agents.

7 Conclusions & Future Work

This chapter presents this thesis' conclusions (**Section 7.1**) as well as possibilities for future work (**Section 7.2**).

7.1 Conclusions

In a world concerned with ecological, social and economic sustainability, in keeping biodiversity, controlling carbon dioxide emissions, managing natural resources exploitation, and feeding our population, among others, agro-ecosystems should be at the core of any approach concerning these problems, since it brings together two fundamental components: the social sub-system and the natural sub-system. Therefore, any attempt to study an agro-ecosystem should jointly consider human actions and natural resources. **Chapter 2** showed that the multi-agent systems approach was a natural and powerful metaphor for modeling such a complex system, with the benefit of setting the ground for the construction of a computational model, which can simulate the evolution of the agro-ecosystem over time, synthesizing complex knowledge, envisioning possible future scenarios and enabling prospective analysis. This simulation gains even more relevance because this kind of systems cannot be directly manipulated.

A successful modeling and simulation of such a system needs the skills of different people, therefore urging for an interdisciplinary team. These skills include (but are not limited to): social sciences (e.g. sociologists and anthropologists) for planning and conducting workshops, interviews and any activity involving producers; natural sciences (e.g. agronomist, ecologists and extensionists) for understanding natural processes, and the dynamics that take place when humans manipulate an ecosystem; economics (e.g. economists) for understanding one of the major forces that play in any human activity: money; and computer science (e.g. software developers and modelers) for putting all this information together into models (e.g. following the UML formalism), and later into computational models (i.e. software) that allow these models to take life.

This thesis makes a contribution by proposing the first AOSE methodology (in its early stages, therefore referred to as *methodological framework* in **Chapter 4**), that guides a software developer into constructing agro-ecosystem simulations using the multi-agent systems approach.

Even though many AOSE methodologies already exist (as shown in **Chapter 3**), they are general-purpose. This implies that they do not consider the *simulation* of a multi-agent system and neither the specificities/features of agro-ecosystems. There is a difference in developing software for, say, an Internet robot, than for simulation purposes, even though both are cases of multi-agent systems. There is also a big gap when trying to use these general-purpose AOSE methodologies for a specific context such as an agro-ecosystem, especially when representing the features discussed in **Chapter 5**.

In the current context where software developers (and programming skills in general) are scarce, a methodology that leverages the object-oriented background of a developer, defining specific steps and artifacts for this niche of agro-ecosystem simulations, should lower the learning curve and ease the burden over programmers to get productive as quickly as possible.

The case study presented in **Chapter 6** takes this methodological framework one step further by applying it to a very relevant problem: the draught phenomena in the Uruguayan basalt region. This allowed to show in a real-world situation the applicability and usefulness of the methodological framework, at the time that it helped in studying and understanding the draught strategies of basalt producers and their consequences over cattle, and their financial balance of adopting them.

Finally, an extract of the work included in this thesis was submitted as a paper to the 2011 International Conference on Agents and Artificial Intelligence to be held in Rome [ICAART, 2011]. The paper was accepted as a poster in the conference and can be found in **Appendix D**.

The results obtained so far are very promising and encourage us to continue working and investigating on these topics. Therefore, the following section introduces several future work related to that already done in this thesis.

7.2 Future Work

The future work proposed in this section is divided into several points: those concerning the methodological framework, those concerning the case study, and those concerning the tools that support it. Some of the points here discussed were already aforementioned in **Chapter 6**.

Future work concerning the methodological framework:

- **Thorough AOSE Methodology Comparison**: even though **Chapter 3** presented several general-purpose AOSE methodologies, a thorough comparison between these and the methodological framework was not given. Such a comparison could give the latter new elements not currently considered. It is noteworthy to remember that the methodological framework presented in **Chapter 4** was originated from another methodology that summarized others. This may have led to ignoring details from these already existing AOSE methodologies that could be useful in the context of agro-ecosystem simulation.
- **Propose a Complete Software Development Methodology**: as it was mentioned several times, the methodological framework is a guideline that only focuses on the software development activities and artifacts, but completely ignores other relevant aspects such as resource, time and risk management, budget and effort estimation, team work, and testing. Proposing a truly comprehensive software development methodology from this software engineering point of view may benefit from a more widely adoption, since it would cover all major aspects of the development, and not only the software product.
- **Relation to Other Disciplines**: somehow related to the previous point, it would be interesting to further analyze which disciplines are indeed needed to successfully achieve an agro-ecosystem simulation using this methodology. For example, what profiles are needed, with what skills, and how they should interact with each other.
- **Further Investigate Input Parameters**: as stated in **Chapter 6**, input parameters influence the development at several points, and they are mostly left behind by the proposed methodological framework. This should encourage to further investigate their role within the methodology, along with guidance on how to define, represent, initialize and use them.
- **Further Investigate Complex Adaptive Systems Support**: even though the case study is far from being regarded as 'simple', it cannot be regarded as a case of complex adaptive system as introduced in **Section 2.1**. Features such as emergence, self-organization and learning do not appeared in the case study of **Chapter 6**. This leaves room for further investigations on whether the proposed methodological framework can effectively support the development of truly complex and adaptive systems.
- **Define Agro-Ecosystem Patterns**: just as design patterns can be applied when designing an OOP system, the agro-ecosystem simulation taken as a specific domain can also give place to patterns (e.g. already defined pieces of UML diagrams including some constructs to represent widely used features or configurations). Moreover, an agro-ecosystem catalog of patterns may be constructed and even implemented over certain agent-based simulation platform (as CORMAS), in order to let the user drag and drop these pre-defined pieces of models and software into a new simulation project.
- **Use of other time management:** as **Section 4.1** established, the methodological framework assumes a time-stepped simulation. Nevertheless, other forms of time management are available, most notably discrete-event, which is the approach of the MIMOSA tool [MIMOSA, 2010].
- **Use of a Domain Specific Language (DSL)**: this thesis assumed the use of plain UML diagrams in order to facilitate developer's uptake. Nevertheless, the use of more specific languages such as AML could benefit the methodological framework by giving more accurate constructs (e.g. being able to natively model an agent or a resource on a diagram). Furthermore, the use of DSLs would allow modeling not just an "agent" natively but a "farmer", a "strategy" and any of the agro-ecosystems' features introduced in **Chapter 5**. This would also relate to the use of a simulation tool that natively supports such DSLs constructs (refer to the point "**DSL Simulation Tool"** below).

Future work concerning the case study:

- **Define Agent Interaction**: Step 5 of the methodological framework did not get the chance of being applied to the case study. Therefore, a new case study should be defined (or at least the actual one be modified) in order to analyze how this step really contributes to the problem of defining and representing which agents communicate to which others, what they exchange, when and how.
- **Use of other Agent Architectures**: the case study used a Reactive architecture, and similarly to the previous point, this did not give the chance to see how the methodological framework supports the use of other agent architectures, so new case studies should be defined that tackle this.
- **Use a General AOSE Methodology**: in order to better understand and compare the methodological framework presented in this thesis with already existing AOSE methodologies, these could be applied to the simulation of an agro-ecosystem. This comparison should show the benefits of counting with a specialized methodology for agro-ecosystem simulation.

Future work concerning the tools that support the methodological framework:

- **Use of other Simulation Software**: as it is clear from the case study, this thesis used the CORMAS simulation platform. Even though it proved to be extremely useful and powerful, no alternatives where analyzed. It would be important to study other agent-oriented simulation platforms, discuss which of them could be more aligned with the proposed methodological framework, and use a couple of these in case studies, maybe even implementing this very same case study on other simulation platforms and compare the results[19].
- **Other AOSE Tools**: related to the first point (**Thorough AOSE Methodology Comparison**) it would be worth to study which tools do these already existing and general purpose AOSE methodologies use as assistants for the actual programming and/or modeling. These tools could be even integrated to mainstream Integrated Development Environments such as Eclipse [Eclipse, 2010] or NetBeans [NetBeans, 2010] or be standalone tools that were built specially for a certain AOSE methodology. The proposed methodological framework could incorporate new ideas from these and even use them if possible.
- **New Tool for the Proposed Methodological Framework**: apart from studying other tools as the previous point proposes, an analysis could be made of which tools would best support the current methodological framework (or its "full-size" one as already proposed). Such a tool could be imagined to be designed from zero to be completely aligned to support the steps here presented. It would be a custom-made tool (or set of tools) especially designed and implemented, tailored for the proposed methodological framework. This could facilitate even more the adoption of this methodology for standard OOP software developers.
- **DSL Simulation Tool**: relating the previous point with the point "**Use of a Domain Specific Language**" a specific simulation tool could be developed that natively maps the domain-specific constructs (agent, resource, crop) to programming code. This implies developing a library of classes that serve as base classes for all of these domain-specific constructs, along with a graphical tool that would allow the user/programmer to declaratively define/draw using the DSL and the tool would produce the corresponding code. To this end, several undergraduate projects have been conducted using CORMAS and Eclipse as the programming environments [FING1, 2008; FING2, 2009; FING3, 2010].

[19] Railsback et al., (2006) have described a simple model, called "Stupid Model" which has been used as a benchmark model to compare existing agent-based simulation platforms.

References

Altieri, M. 1987. *Agroecology: The Science of Sustainable Agriculture*. Westview Press.

AML, 2010. Website: www.whitestein.com/aml Last visited: December 2010.

AOSE, 2010. Website: http://grasia.fdi.ucm.es/aose08/index.html Last visited: December 2010.

Axelrod, R. 1984. *Evolution of cooperation*. Basic Books, New York, New York, USA.

Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

Bernon, C., Gleizes, M.-P., Picard, G., & Glize, P. 2002. *The ADELFE methodology for an intranet system design*. In P. Giorgini, Y. Lespérance, G. Wagner, & E. Yu (Eds.), Proceedings of Agent-Oriented Information Systems, AOIS-2002: 1-15.

Birtwistle, G.M., O.-J. Dahl, B. Myhrhaug and K. Nygaard. 1973. *SIMULA Begin*, AUERBACH Publishers Inc.

Booch, G. 1983. *Software Engineering with Ada*. Benjamin-Cummings Publishing Co., CA, USA.

Booch, G. 1995. *Object Solutions: Managing the Object-Oriented Project*. Reading, MA. Addison-Wesley.

Bousquet, F., Le Page, C. 2004. *Multi-agent simulations and ecosystem management: a review*. Ecological Modeling 176:313-332.

Bousquet, F., Cambier, C., Morand, P. 1994. *Distributed artificial intelligence and object-oriented modeling of a fishery*. Math. Comp. Model. 20: 97-107.

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

Briggs, D. J., Courtney, F. M.1985. *Agriculture and Environment*. Longman Press, London.

Brooks, R. 1990. *Elephants Don't Play Chess*. Robotics and Autonomous Systems 6: 3-15.

Budd, T. 2001. *Understanding Object-Oriented Programming with Java*. Pearson Education.

Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., & Massonet, P. 2001. *Agent-oriented analysis using MESSAGE/UML*. In M. Wooldridge, G. Wei, & P. Ciancarini (Eds.), *Agent-oriented software engineering II* (p. 119-135). LNCS 2222. Berlin: Springer-Verlag.

CIRAD, 2010. Website: www.cirad.fr Last visited: December 2010.

Coleman, G., Verbruggen, R. 1998. *A Quality Software Process for Rapid Application Development*. Software Quality Control 7, 2: 107-122.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., & Gilchrist, H. 1994. *Object-oriented development. The fusion method.* Englewood Cliffs, NJ: Prentice Hall.

CORMAS, 2010. Website: http://cormas.cirad.fr Last visited: December 2010.

Cossentino, M. 2005. *From requirements to code with the PASSI methodology*. In B. Henderson-Sellers & P. Giorgini (Eds.), *Agent-oriented methodologies* (Chapter 4). Hershey, PA: Idea Group.

CSIRO, 2008. *Complex or just complicated: what is a complex system?* CSIRO fact sheet. URL: www.csiro.au/resources/AboutComplexSystems.html Last visited: November 2008.

Davis, R. 1980. *Reports on the workshop on Distributed AI*. ACM Sigart Newsletter 73: 42-52.

Dawn, P., Hessl, A., Davis, S. 2008. *Complexity, land-use modeling, and the human dimension: Fundamental challenges for mapping unknown outcome spaces*. Geoforum, Volume 39 Issue 2: 789-804.

Deffuant G., Weisbuch G., Amblard F. and Faure T. 2003. *Simple is beautiful ... and necessary*. Journal of Artificial Societies and Social Simulation, vol. 6, no. 1 http://jasss.soc.surrey.ac.uk/6/1/6.html

DeLoach, S.A. 1999. *Multiagent Systems Engineering: A methodology and language for describing agent systems*. In Proceedings of the First International Bi-conference Workshop on Agent-Oriented Information Systems (AOIS '99), Seattle, USA.

Durkheim, E. 1979. *Suicide: A Study in Sociology*. Free Press, NJ.

Eclipse, 2010. Website: www.eclipse.org Last visited: December 2010.

Ellen, R. 1982. *Environment, Subsistence and Systems*. Cambridge University Press, New York.

Ferber, J. 1995. *Les Systèmes Multi-Agents. Vers une Intelligence Collective*. InterEditions, Paris.

Ferber, J. 1999. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, Reading, MA.

FING1, 2008. *Domain Specific Language for Multi-Agent System Simulations (DSL4MASS)*. Diego Salido & Horacio Blanco. Proyecto de Grado, Facultad de Ingeniería, UdelaR. Uruguay.

FING2, 2009. *Framework de Simulación Basado en Sistemas Multi-Agente*. Diego Guerra, Yanella Sanchez & Paola Iraola. Proyecto de Grado, Facultad de Ingeniería, UdelaR. Uruguay.

FING3, 2010. *Framework de Simulación en Java Basado en Sistemas Multi-Agente.* Ramiro Simón, Bruno Caetano, Carlos Cristoforone & Magela Cairo, Proyecto de Grado, Facultad de Ingeniería, UdelaR. Uruguay.

FIPA, 2010. Website: www.fipa.org Last visited: December 2010.

Fowler, M. 1996. *Analysis Patterns: Reusable Object Models*. Addison-Wesley.

Fowler, M. 2003. *UML Distilled*. Third Edition. Addison-Wesley.

Galler, H.P. 1997. *Discrete-time and continuous-time approaches to dynamic microsimulation reconsidered*. Online Technical Paper - TP13. October 1997, NATSEM, University of Canberra

Gamma, E., Helm R., Johnson R., Vlissides J. 1994. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley.

Gardner, M., 1970. *The fantastic combinations of John Conway's new solitaire game ´life´*. Scientific American 223: 120-123.

Gliessman, S. R. 1997. *Agroecology: ecological processes in sustainable agriculture*. Ann Arbor Press

Gosling, J., Joy, B., Stelle, G., Bracha, G. 2005. *The Java Language Specification*. Third Edition. Addison-Wesley.

Grimm V, Railsback SF. 2005. *Individual-based Modeling and Ecology*. Princeton University Press, Princeton N.J., 428 pp.

Hecht, S. 1987. *The Evolution of Agricultural Thought*. In [Altieri, 1987].

Henderson-Sellers, B., Giorgini, P. (Eds.) 2005. *Agent-oriented Methodologies*. Hershey, PA: Idea Group.

Hogeweg, P., Hesper, B. 1983. *The ontogeny of the interaction structure in bumble bee colonies: a mirror model*. Behavioral Ecological Sociobiology 12: 271-283.

Huston, M., DeAngelis, D., Post, W. 1988. *New computer models unify ecological theory*. Bioscience 38: 682-691.

ICAART, 2011. Website: www.icaart.org Last visited: December 2010.

Iglesias, C.A., Garijo, M., Gonzalez, J.C., & Velasco, J.R. 1996. *A methodological proposal for multiagent systems development extending CommonKADS*. In *Proceedings of 10th KAW*, Banff, Canada.

INIA, 2010. Website: www.inia.org.uy Last visited: December 2010.

IPA, 2010. Website: www.planagropecuario.com.uy Last visited: December 2010.

JAAMAS, 2010. Website: www.ifaamas.org Last visited: December 2010.

Jennings, N. 2001. *An agent-based approach for building complex software systems*. In Communications of the ACM. Volume 44, Issue 4.

Kay, A. 1993. *The Early History of Smalltalk*. ACM.

Kinny, D., Georgeff, M., & Rao, A. 1996. *A methodology and modeling techniques for systems of BDI agents*. Technical Note 58, Australian Artificial Intelligence Institute, also published in *Proceedings of Agents Breaking Away, the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)* (pp. 56-71). Springer-Verlag.

Kruchten, P. *The Rational Unified Process: An Introduction*. Third Edition. Addison-Wesley.

Kruchten, P. 1999. *The Rational Unified Process: An Introduction*. First Edition. Addison-Wesley.

Langton, C. 1988. *Artificial Life*. In: Langton, C. (ed.), *Artificial Life*. Addison-Wesley, pp. 1-47.

Lansing, J. S., Kramer, J. N. (eds.) 1994. *Emergent Properties of Balinese Water Temple Networks: Coadaptation on a Rugged Fitness Landscape*. In: Langton, C. (ed.) Artificial Life III. Addison-Wesley, Santa Fe.

Lind, J. Website 2008. Website: www.agentlab.de Last visited: December 2010.

Matthews, R., 2006. *The People and Landscape Model (PALM): Towards full integration of human decision-making and biophysical simulation models.* Ecol. Model. 194, 329–343.

McCarthy, J., Minsky, M. L., Rochester, N., Shannon, C. E. 1955. *A proposal for the Dartmouth summer research project on Artificial Intelligence*. Dartmouth College, USA.

Mellor, S., Balcer, M. 2002. *Executable UML: A foundation for model-driven architecture*. Addison-Wesley.

Michener, W. K., Baerwald, T. J., Firth, P., Palmer, M. A., Rosenberger, J. L., Sandlin, E. A., Zimmerman, H. 2001. *Defining and Unraveling Biocomplexity*. Bioscience, Volume 51 Number 12: 1018-1023.

Microsoft's .NET's 2008. Website: www.microsoft.com/NET/ Last visited: December 2010.

Miller, J.H., Page, S.E. 2007. *Complex adaptive systems: an introduction to computational models of social life*. Princeton University Press.

MIMOSA, 2010. Website: http://mimosa.sourceforge.net Last visited: December 2010.

NetBeans, 2010. Website: www.netbeans.org Last visited: December 2010.

Object Management Group Website. 2008. Website: www.omg.org Last visited: December 2010.

Odum, E. P. 1969. *The strategy of ecosystem development*. Science 164: 262:270.

Odum, E. P. 1984. *Properties of Agroecosystems*. In: *Agricultural Ecosystems*. Lowrance et al. (eds), Wiley Interscience, New York.

Padgham, L. & Winikoff, M. 2002. *Prometheus: A methodology for developing intelligent agents*. In F. Giunchiglia, J. Odell, & G. Weiss (Eds.), *Agent-oriented Software Engineering III Proceedings of the Third International Workshop on Agent-Oriented Software Engineering (AAMAS'02)* (pp. 174-185). LNCS 2585.

Parker, D.C., Manson, S.M., Janssen, M.A., Hoffmann, M.J., Deadman, P. 2003. *Multi-Agent Systems for the Simulation of Land-Use and Land-Cover Change: A Review*. Annals of the Association of American Geographers, Volume 93, Issue 2: 314-337.

Parrott, L. 2002. *Complexity and the limits of Ecological Engineering*. In: American Society of Agricultural Engineers, Volume 45, Issue 5: 1697-1702.

Pavón, J., Gomez-Sanz, J., & Fuentes, R. 2005. *The INGENIAS methodology and tools*. In B. Henderson-Sellers & P. Giorgini (Eds.), *Agent-oriented methodologies* (Chapter 4). Hershey, PA: Idea Group.

Perez, P., Batten, D. (editors). 2006. *Complex Science for a Complex World, Exploring Human Ecosystems with Agents.* The Australian National University Press, Australia.

Petra, P., Tolksdorf, R., Zambonelli, F. 2003. *Lecture notes in Artificial Intelligence*. In Engineering, Societies in the Agents World III. Springer.

Pfleeger, S. L. Atlee, J. 2006. *Software Engineering. Theory and Practice*. Third Edition. Pearson.

Rammel, C., Stagl, S., Wilfing, H., 2007. *Managing complex adaptive systems – A co-evolutionary perspective on natural resource management*. Ecol. Econ. 63, 9–21.

Railsback, S. F., Lytinen, S. L., & Jackson, S. K. 2006. *Agent-based simulation platforms: review and development recommendations*. *Simulation*, 82(9): 609-623.

Reynolds, C. 1987. *Flocks, herds and schools: a distributed behavioral model*. Comput. Graph. 21: 25-34.

Royce, W. 1970. *Managing the Development of Large Software Systems*. Proceedings on IEEE WESCON 26: 1-9.

Rumbaugh, J., Jacobson, I., Booch, J. 1998. *The Unified Modeling Language Reference Manual*. Addison-Wesley.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. 1991. *Object-oriented modeling and design*. Englewood Cliffs, NJ: Prentice-Hall.

Shelling, T. C. 1971. *Dynamic models of segregation*. Journal of Mathematical Sociology 1: 143-186.

Shelling, T. C. 1978. *Micromotives and macrobehavior*. W. W. Norton, New York, New York, USA.

Senge, P. 1994. *The fifth discipline: the art and practice of the learning organization*. Doubleday Business, NY.

Shoham, Y. 2010. Website: http://hci.stanford.edu/courses/cs547/abstracts/94-95/950113-shoham.html Last visited: December 2010.

Smith, A. 1776. *An Inquiry into the Nature and Causes of the Wealth of Nations*. Edited by Andrew Skinner. New York: Pelican Books, 1970.

Software Industry Report. 1997. *Corporate America plans to spend big on object-oriented software*. Software Industry Report January 1997. Source: http://www.allbusiness.com/technology/computer-software/603292-1.html Last visited: December 2010.

Sommerville, I. 2006. *Software Engineering*. Eighth Edition. Pearson.

Stepp, J.R., Jones, E.C., Pavao-Zuckerman, M., Casagrande, D., Zarger, R.K. 2003. *Remarkable properties of human ecosystems*. In: Conservation Ecology, Volume 7 Issue 3: 11.

Stroustrup, B. 1991. *What is 'Object-Oriented Programming'?* AT&T Bell Laboratories Paper, Murray Hill, New Jersey. Source: www.research.att.com/~bs/whatis.pdf Last visited: December 2010.

Stroustrup, B. 1994. *The Desgin and Evolution of C++*. First Edition. Addison-Wesley.

Taveter, K. & Wagner, G. 2005. *Towards radical agent-oriented software engineering processes based on AOR modelling*. In B. Henderson-Sellers & P. Giorgini (Eds.), *Agent-oriented methodologies* (Chapter 10). Hershey, PA: Idea Group.

Travers, M. D. 1996. *PhD Dissertation*. Massachusetts Institute of Technology.

Tran, Q.N., Low, G. 2008. *MOBMAS: A methodology for ontology-based multi-agent systems development*. In Information and Software Technology, 50: 697-722.

Turner, M. 2006. *Microsoft Solutions Framework Essentials*. Microsoft Press.

UML, 2010. Website: www.uml.org Last visited: December 2010.

Von Bertalanffy, L. 1968. *General System Theory*. George Braziller, NY.

Wooldridge, M. 2002. *An Introduction to Multiagent Systems*. Wiley & Sons.

Wooldridge, M. Website 2008. Lecture Notes on Introduction to Multiagent Systems Course based on based on [Wooldridge, 2002]. Website: www.csc.liv.ac.uk/~mjw/pubs/imas/teaching.html Last visited: December 2010.

Wooldridge, M., Jennings, N. R. 1995. *Intelligent agents: theory and practice*. The Knowledge Engineering Review 10(2): 115-152.

Wooldridge, M., Jennings N.R., & Kinny, D. 2000. *The Gaia methodology for agent-oriented analysis and design*. Journal of Autonomous Agents and Multi-Agent Systems, 3: 285-312.

Yu, E. 1995. *Modelling strategic relationships for process reengineering*. PhD Thesis, University of Toronto, Department of Computer Science.

Appendix A: State-of-the-Art of MAS

This appendix gives a more in-depth state-of-the-art of the topics presented in Chapter 2. Section 1 presents fundamental concepts and principles related to complex systems which are the general context for agro-ecosystems presented in Section 2. Section 3 introduces the agent-based modeling approach, and Section 4 shows the applicability of the agent-based modeling approach to complex systems in general and to agro-ecosystems in particular. Section 5 gives some basic definitions of software engineering concepts, and Section 6 presents a comparison between the Object-Oriented Programming and the Agent-Oriented Programming paradigms.

1. Complex Systems

In natural sciences and computer science, the most common approach to understand or analyze a system is based on a decomposition of the system into its elementary parts and the isolated and in-depth study of these in order to understand the whole, i.e. the reductionism of Descartes. A good example, very well-known in computer science, is the 'divide and conquer' approach, which suggests to decompose the original system (or problem) into smaller and simpler parts, and to understand (or solve) each individual part and finally add up all of these partial understandings (or partial solutions) in order to achieve the complete understanding of the system (or a complete solution). Early notions, such as modularity (in both, data structures and functionality) and object orientation also correspond to this idea.

The systemic approach (also called 'systems approach', 'holistic approach' or 'systems thinking') starts by first examining and understanding the relations between the different elements of the system. As the very term 'holistic' suggests, the idea is to never forget the system as a whole, for what it is essential to pay constant attention to the interrelations between the elements. The systemic approach is a general theory since its principles can be applied to any discipline or area. Its origins go back to half of the 20th century with the work of the Austrian biologist Ludwig von Bertalanffy [Bertalanffy, 1968]. One of the most common ways for addressing the systemic approach is to say that 'the whole is more than just the sum of its parts', or in the words of [Miller and Page, 2007]: *"The field of complex systems challenges the notion that by perfectly understanding the behavior of each component part of a system we will then understand the system as a whole"*.

Both complicated and complex systems are composed of a large number of interacting elements, but two properties set a complex system apart from one that is merely complicated: emergence and self-organization. Emergence is the appearance of behavior that could not be anticipated from the knowledge of the parts of the system alone [CSIRO, 2008]. Moreover, the newly emerged properties can in turn feedback to the original lower-level entities, entering a feedback loop where each element (micro/marco level) interacts. Both positive and negative feedback loops are possible; positive feedback loops lead to expansion of a given behavior or dynamics, and negative feedback results in mitigation and control. Self-organization means that there is no external controller or planner engineering the appearance of the emergent features; they appear spontaneously [CSIRO, 2008]. The motivation for studying complex systems is that many of current opportunities and challenges (globalization, sustainability, terrorism, epidemics, climate change) are complex. Each of these domains consists of a set of diverse entities and actors that dynamically interact, and are immersed in a sea of feedback [Miller and Page, 2007]. **Figure A.1** shows a schematic representation of a complex system based on [Parrott, 2002].

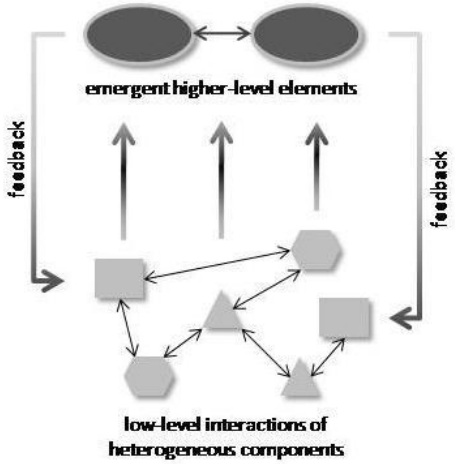


Figure A.1: Schematic Conceptual Model of a Complex System.

A key feature of real systems that has proved to be essential in the appearance of rich emergent features is local interaction. In other words, elements of a system only interact with their neighbors [CSIRO, 2008]. Well-known and simple rules in the micro-level can make the emergence of system-level phenomena, as in the case of John Conway's Game of Life [Gardner, 1970] where a simple set of rules about when a certain pixel is considered 'alive' (black pixel) or 'dead' (white pixel) depending on the number of 'alive' neighbors, gives rise to a perfectly clear pattern of 5-pixel 'flying gliders' that flies across the screen with a perfectly defined trajectory[20], which is not predictable from the simple set of rules. Another example of emergence comes from Durkheim's study of suicide[21] [Durkheim, 1979] where a process that seems to be governed by chance, when viewed at the level of individuals, turns out to be strikingly predictable at the level of society as a whole [Perez and Batten, 2006].

Probably the earliest reference in history to complexity (involving social sciences) and emergence is Adam Smith's Wealth of Nations [Smith, 1776] book, where he describes the concept of an 'invisible hand' leading collections of self-interested agents into well-formed structures that are no part of any single agent's intention.

Emergent properties and complexity usually arise when the relations among the elements of the system are not linear. This means that the behavior of a single element is not the result of a linear combination of the individual behaviors of related elements, leading to a non-linearity. Another source of complexity appears when there are time and scale differences between cause and effect. That is, if an action in certain level of organization or hierarchy at a certain time ends up having its effect over a long period of time and over different levels of organization or hierarchies (different scales), then the understanding of the cause→effect relationship vanishes [Senge, 1994].

Even though several renowned scientists (for example in the Santa Fe Institute[22]) are behind the idea that very different realities and problems can share some common principles (described by complex systems theory), critics also disbelieve such generally-applicable principles (as it is shown in the article titled "From Complexity to Perplexity" from John Horgan, Scientific American senior writer, June 1995[23]).

Complex Adaptive Systems' (CAS) are defined as systems that are capable to adapt and self-organize in response to perturbations or distortions in the environment, or by the result of certain interrelations between the elements. System adaptation is ultimately concerned with the adaptation of each individual element of the system, since there is no centralized control and therefore no single 'object' that represents the entire system. This also relates to the concept of co-evolution of the different elements or parts of the system, which by means of their interrelations evolve their behavior over time, adapting themselves to new situations [Rammel et al., 2007].

Adaptation of a single element is commonly associated with the display of alternative behaviors. Learning can be defined as a mechanism to attain the ability to exhibit different behavior over time.

[20] An on-line interactive application is freely available at: http://www.bitstorm.org/gameoflife/
[21] "It is impossible to predict which of them are likely to kill themselves. Yet the number of Parisians who commit suicide each year is even more stable than the general mortality rate." [Durkheim, 1979]
[22] The Santa Fe Institute coined the term 'complex adaptive systems' (www.santafe.edu)
[23] Available at: http://www.econ.iastate.edu/tesfatsi/hogan.complexperplex.htm

The larger the capacity to learn, the more capable it will be to adapt to new conditions. One well-known researcher on adaptive systems is John Holland, one of the first PhDs ever in computer science, who created the genetic algorithms which aim at evolving generation over generation of a certain population, by mutations or cross-breeding between individuals that present the best survival capacities (measured for example, by a fitness function) and constitute an example of a computer science adaptation strategy. Adaptation and learning mechanisms generally increase the complexity of the whole system, since the behavior of the elements is able to change and evolve over time through adaptation processes.

2. Agro-Ecosystems

This section introduces the notion of agro-ecosystem and related concepts. Also, agro-ecosystems are presented as a special case of complex systems. Finally, the importance of modeling and simulation of agro-ecosystems is highlighted.

2.1 What is an Agro-ecosystem?

An agro-ecosystem is the human manipulation and alteration of ecosystems for the purpose of establishing agricultural production [Gliessman, 1997]. Agro-ecosystems result from the interplay between endogenous biological and environmental features of the agricultural fields, and exogenous social and economic factors, and are delimited by arbitrarily chosen boundaries. They are semi-domesticated ecosystems that fall on a gradient between ecosystems that have experienced minimal human impact, and those under maximum human control [Hecht, 1987].

According to [Odum, 1984] the four major characteristics of agro-ecosystems are:

- They include **external sources** of energy like human, animal or fuel energy to enhance productivity of particular crops;
- **Diversity** may be reduced compared with natural ecosystems (those with no human intervention);
- The dominant animals and plants are under **artificial selection** rather natural selection; and
- The **system controls** are external rather than internal via subsystem feedback, in the sense that the natural resources are no less dependent on natural factors because of human intervention.

Even though this is a good starting point according to [Hecht, 1987] it does not reflect agro-ecosystems that can be relatively diverse and its lack of attention to the social determinants of agriculture limits its explanatory power. Agricultural systems are human artifacts, and the determinants of agriculture do not stop at the boundaries of the field. Agricultural strategies respond not only to environmental, biotic[24], and cultivar constraints, but also reflect human subsistence strategies and economic conditions [Hecht, 1987; Ellen, 1982]. This stresses the importance of social factors like labor availability, access and conditions of credit, subsidies, perceived risk, price information, association obligations, family size, and access to other forms of livelihood, which are often critical to understanding the logic of a farming system [Hecht, 1987].

An agro-ecosystem thus, has physical parts with particular relationships (the *structure* of the system) that together take part in dynamic processes (the *function* of the system) [Gliessman, 1997]. The structure can be viewed as organized in several levels, ranging from individual elements, such as organisms or crops, up to regions, landscapes or entire countries. This organization also allows the analysis of the different scales present in an agro-ecosystem, as well as the relations between scales. The functioning of the agro-ecosystem refers to the dynamic processes occurring within ecosystems: the flow of matter and energy and the interactions and relationships of the organisms and materials in the system.

Any system uses its processes and resources to convert its inputs into its outputs. Concerning the resources commonly found in agro-ecosystems, [Norman, 1979] suggests the following classification:

- **Natural resources**: the given elements of land, water, climate and natural vegetation that are exploited by the farmer.
- **Human resources**: the people who live and work within the farm and use its resources for agricultural production, based on their traditional or economic incentives.

[24] Associated with or derived from living organisms (English On-Line Dictionary).

- **Capital resources**: the goods and services created, purchased, or borrowed by the people associated with the farm to facilitate their exploitation of natural resources for agricultural production.
- **Production resources**: the agricultural output of the farm, such as crops and livestock. These become capital resources when sold, and residues (crops, manure) are nutrient inputs reinvested in the system.

Several of all the previous concepts are summarized in **Figure A.2**, based on [Altieri, 1995] and [Briggs and Courtney, 1985]:

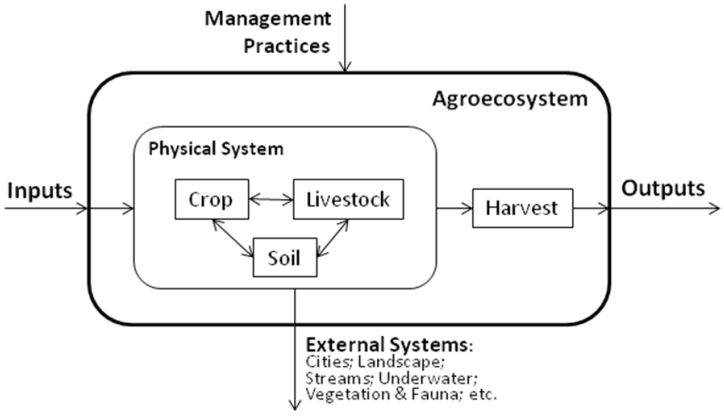


Figure A.2: Representation of an agro-ecosystem.

It is worth noting that human intervention is represented by the inputs, but mainly by the management practices executed by farmers.

Other concepts that are also commonly associated to agro-ecosystems include natural processes (like erosion), the landscape to which the agro-ecosystem belongs, market prices and prices evolution (regarding for example international crop prices), and government policies (which may affect virtually any part of the agro-ecosystem).

Finally, the main differences between natural ecosystems and agricultural ecosystems can be, according to [Gliessman, 1997] and closely related to the characteristics of agro-ecosystems of Norman, summarized[25] as:

- **Differences in energy flow**: energy is taken away at each harvest rather than stored in biomass;
- **Differences in nutrient cycling**: recycling of nutrients is minimal in most agro-ecosystems and considerable quantities are lost in harvest or as a result of erosion;
- **Differences in population regulation mechanisms**: due to simplification (reduction of diversity) populations (crops or animals) are rarely self-reproducing or self-regulating, rather, human inputs (seeds, control agents) determine population sizes; and
- **Differences in stability**: due to their reduced structural and functional diversity, agro-ecosystems have much less resilience than natural ecosystems.

Figure A.2 and the previous concepts suggest that agro-ecosystems can be seen as a case of complex systems. The following section looks further into this idea.

2.2 Agro-ecosystems as Complex Adaptive Systems

Any ecosystem by itself can already be considered as a case of complex adaptive system, considering its various components, organization levels, micro and macro interactions and their feedbacks, and even more if including a social subsystem (with heterogeneous behaviors) as in the case of agro-ecosystems.

The interactions between the social and natural subsystems inside an agro-ecosystem, such as farmers' practices affecting natural resources, are examples of local interactions at a small scale which may produce negative or positive feedbacks affecting in turn the decision-making of social actors. Repeating these interactions on a daily basis can produce emergent properties and new organizations

[25] Other structural and functional differences between natural ecosystems and agro-ecosystems can be found in Gliessman, 1997 and Odum, 1969 (also in http://www.agroecology.org/Principles_Eco.html).

across the agro-ecosystem in the long-term, like a change in soil quality, as a long-term consequence of the adoption of certain management practice or the emergence of a certain land-use pattern.

Furthermore, according to [Dawn et al., 2008] the sources of complexity in coupled human-natural systems (as is the case of agro-ecosystems) arises mainly from temporal, spatial and scale mismatches between actions and their impacts (similarly to what Senge proposes), which occur due to the indirect and imperfect nature of these interactions between human actions and their impact on the environment.

Aside from natural processes, also the interactions between the social actors can lead to complex behaviors like, for example, communications between neighboring producers which can affect each producer's performance (e.g. knowing about certain new market because of participation in an organization can lead to improvements in the income of those producers that receive the correct information at the right moment).

[Gliessman, 1997] addresses some of these issues and states what could be considered the ultimate emerging property of an agro-ecosystem:

> *"An important characteristic of ecosystems is that at each level of organization properties emerge that were not present at the level below. These emergent properties are the result of the interaction of component parts of that level of ecosystem organization. In an agro-ecosystems context, this principle means in essence that the farm is greater than the sum of its individual crop plants. Sustainability can be considered the ultimate emergent quality of an ecosystem approach to agriculture."*

On the other hand, the sources of adaptation come from the capacity of humans to adapt their behavior by learning from their experience, and from the capacity of this environment to adapt to new conditions and constraints. According to [Perez and Batten, 2006] these kinds of coupled systems, involving people, other living entities, an environment, information exchange and the co-evolution of all of these things over time, are inherently complex and adaptive due to the ability of human beings to switch from rational to deductive reasoning. This flow of information, besides the matter and energy flows, is also referred to by [Stepp et al., 2003].

Recently a new term has been coined: biocomplexity. This is defined as properties emerging from the interplay of behavioral, biological, chemical, physical, and social interactions that affect, sustain or are modified by living organisms, including humans [Michener et al., 2001]. One of the very first uses of the term which triggered further research was adopted by a new US National Science Foundation[26] Priority Area on Biocomplexity, who started funding programs on 1999 with a very broad sense of biocomplexity, but then moved towards more specific issues, being one of them the 'Dynamics of Coupled Natural and Human Systems'.

Finally, as stated in [Michener et al., 2001] two particularly salient features of biocomplexity are that 1) it arises as temporal, conceptual and spatial boundaries are breached; and 2) the system may exhibit emergent or unexpected properties (i.e. the behavior of the whole is often not predictable based on a study of its component parts).

This makes more evident the relation between agro-ecosystems as complex systems and biocomplexity as a new area, as well as demonstrates current interest in research over these topics by providing extensive funding (approximately 125 million dollars on the first three NSF Biocomplexity Phases [Michener et al., 2001]).

2.3 The Importance of Simulation in Agro-ecosystems

Since these coupled human-natural systems cannot be manipulated and tested as other systems, due to scale and resource difficulties (setting up farms managed by people over decades just to see the effects) the possibility of simulating them is crucial.

The objective of simulating these kind of systems is not trying to figure out exactly what will happen in the future if the present conditions are X, the behavior of each part is Y and the evolution of certain parameters is Z. In contrast, simulation in these contexts should have the objective of prospection of scenarios were the interest is not in the 'fortune-teller' features of the simulation but in discovering possible outcomes under certain conditions and being able to easily modify these conditions and check again the outcomes (as in a virtual laboratory). Another objective can be to

[26] NSF's Biocomplexity Priority Area: www.nsf.gov/news/priority_areas/biocomplexity/index.jsp

deepen the understanding (and also possibly learning/teaching) of a coupled human-natural system, since the more micro/macro level study of each part as well as their interactions require an in-depth understanding of them in order to obtain a simulation that is close to reality. Yet other objective can be to explore the consequences of manipulating certain system in order to understand how the different parts will co-exist in the future due to that manipulation (e.g. how does a water shortage affects the productivity of a farm, or how certain economic policy impacts in the long term).

Section 3 presents an approach that can be used to develop such simulations in the form of software systems that will be able to simulate the different parts of an agro-ecosystem and monitor (observe) certain parameters over time in order to analyze their evolution.

3. Agent-Based Modeling

The basic ideas around agent-based modeling will be presented in this section with special care in maintaining enough generality and abstraction, trying to (objectively) introduce the topic without any bias of any special discipline or tendency. Later on, several of these general concepts will be put in the particular context of a certain domain, especially in computer science and agro-ecosystems. First, a definition of agent will be given followed by different classifications of agent environments. From that point, the concept of multi-agent system is introduced, as well as its origins and different applications.

Agent based modeling (ABM) is a computer science approach that enables the simulation of heterogeneous populations of interacting individuals or agents, in many cases in a non-agent **environment**, which can also contain non-agent passive **objects** (commonly known as resources). The agents can exhibit a set of different behaviors, and the selected behavior is dependent on the local interactions with other individuals in their **neighborhood** and the state of the environment, thus the agents may be adaptive. The mechanism of selection of behavior can range from simple procedural logic to highly sophisticated **reasoning**. The repertoire of behaviors can be fixed or extensible, and the latter implies that the agents should be able of **learning**. The intelligence of the agent depends on its abilities to reason and to learn. The adaptive behavioral patterns enable self-organization of the population and can result in emergent phenomena. Consequently, the ABM approach is suitable to address complex adaptive systems. These concepts will be further studied in the following sections.

3.1 Definition of Agent

There is no consensus on the definition of agents, but the most used definitions were proposed by [Wooldridge and Jennings, 1995] and [Ferber, 1999]. These definitions have been used alongside each other in a non-competitive way, partly due to the fact that agent models have been applied to many disciplines ranging from computer science to sociology, which pose different demands on 'attributes of agency' [Wooldridge, 2002]. Moreover, the definitions share common principles.

A definition that supports the computer science focus presented in this thesis is proposed by [Wooldridge, 2008]: "An agent is a computer system that is capable of independent action on behalf of its user or owner (figuring out what needs to be done to satisfy design objectives, rather than constantly being told [what to do]". In this definition, the word 'independent' refers to agent autonomy, capable of acting independently and exhibiting control over its internal state. Thus an agent is a computer system capable of autonomous action in some environment in order to meet its design objectives [Wooldridge, 2002&2008] (Section 3.3 further discusses agents' environments).

According to [Wooldridge, 2002] an intelligent agent is a computer system capable of flexible autonomous action in some environment[27] and proposes the following properties in order to let an agent show intelligent behavior: reactive, proactive and social.

- **Reactive**: A reactive system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful) [Wooldridge, 2002]. Even though a changing environment makes agent design harder, it allows the representation of much more complex (and interesting) situations.
- **Proactive**: Since agents should be able to *do* something according to their design objectives (goals) they should have some way to direct their behavior towards those goals. Proactiveness

[27] One of the best known examples of agents are robots. From industry manufacturing to NASA explorers and soccer-playing, robots can be seen as a "*physical instantiation of an agent*" according to Wooldridge. Being their objective to ensemble a car, collect and analyze rocks or score a goal, being alone or in groups, each of these share, to more or less extent, the properties discussed here.

means generating and attempting to achieve goals, not driven solely by events, as in the case of reactivity, but by actually taking the initiative [Wooldridge, 2002].

- **Social**: The real world is a multi-agent environment. As [Wooldrigde, 2002] states: "we cannot go around attempting to achieve goals without taking others into account". Social ability in agents is the ability to interact with other agents via some kind of agent-communication protocol or language, and perhaps also cooperate or negotiate with others.

In order to formalize some of the previous topics and to simplify future discussions without losing generality, the following concepts are introduced, which were extracted from [Wooldridge, 2002] and [Lind, 2008]. In order to let agents react (or take the initiative) according to changes in the environment, agents must perceive their environment and have some way to act upon it, after reasoning what to do. This leads agent to a Perceive/Reason/Act cycle, shown in **Figure A.3**.

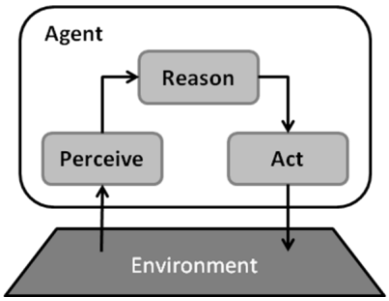


Figure A.3: General representation of an agent with its environment.

Based upon these concepts, the following can be defined:

- **E** the environment, defined as the set of all possible states in which the environment can be;
- **D** the knowledge of the agent, defined as the set of all past perceptions it has received from the environment;
- **A** the actions available for the agent, defined as the set of actions that the agent can do;
- *perceive*: **E** → **T** a function that determines how the environment is perceived by the agent, resulting in the observation **T**;
- *update*: **D** x **T** → **D** a function used to update the agent's knowledge of occurred events;
- *select*: **D** x **T** → **A** a function used to select one action based on the previous knowledge (**D**) and current perceived events (**T**); and
- *act*: **A** x **E** → **E** a function that determines the new state of the environment after executing an action.

Using these definitions an agent can be defined as a 7-tuple <**D**, **T**, **A**, *perceive*, *update*, *select*, *act*>. This is illustrated in **Figure A.4**.

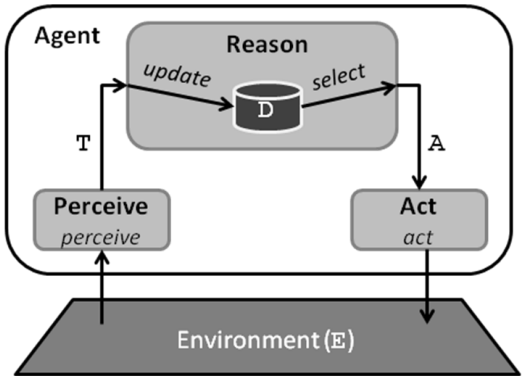


Figure A.4: The Perceive/Reason/Act Cycle Revisited

3.2 Agent Architectures

Since there are multiple ways to achieve this Perceive/Reason/Act cycle, there are accordingly different agent architectures that represent and implement this concept in different ways and to different degrees. The term 'agent architecture' refers to the software architecture for the decision-making agents in the environment, as well as their internal mechanisms and representations [Wooldridge, 2002; Bousquet & Le Page, 2004].

Two issues should be taken into consideration regarding this term: first, that in the context of agents, architectures are generally referred as being abstract (closer to reality) or concrete (closer to implementation); and second, that several authors differ on the classification of agent's architectures.

The following classification of abstract architectures[28] is given by [Wooldridge, 2002]:

- **Deliberative Architectures** (also known as Symbolic or Logical AI) that manages explicit representations of desires (goals), beliefs (what the AI agent knows), intentions (what it wants to do), actions (what it does) and uses abstract reasoning tools. Deductive and deliberative agents appear in these architectures as well as the BDI Architecture (Belief, Desire, Intention)[29];
- **Reactive Architectures** that enable intelligence without having explicit representations, or abstract reasoning, but as an emergent property of certain complex systems. In this kind of architectures the agent has no previous knowledge (**D**) and simply reacts based on a set of rules; and
- **Hybrid Architectures** that include a deliberative as well as a reactive component.

The first type of architecture was the most common in the first stages of Artificial Intelligence which suffered from several drawbacks and difficulties that led to the second type of architecture and to the third type (Section 3.5 shows this historical evolution).

3.3 Different Classifications of Agent's Environments

The environment is what surrounds the agent, where the agent is located, and a means by which the agent receives input from the outside (out of the agent's system). In a context with several agents (as Section 3.4 will present) the environment also allows agents to communicate with each other.

For example, reactivity means that an agent is able to react upon changes on the environment, acting upon it as a reaction to that change. Therefore the relation between the agent and its environment is crucial since it can trigger agent's actions; allow the agent to perceive more or less accurately what happened, etc. The following are some classification of environments (based on [Wooldridge, 2002]):

- **Accessible vs. Inaccessible**: an accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state. If any constraints exist then the environment is called inaccessible. This can be represented imposing restrictions in the *perceive* function by limiting the agent's perception (if **T** < **E**) or not (if **T** = **E**);
- **Deterministic vs. Non-deterministic**: a deterministic environment is one in which any action has a single guaranteed effect –there is no uncertainty about the state that will result from performing an action.
- **Static vs. Dynamic**: a static environment is one that can be assumed to remain unchanged except by the performance of actions by agents. A dynamic one has other processes operating on it and which hence changes in ways beyond the agent's control.
- **Discrete vs. Continuous**: an environment is discrete if there is a fixed, finite number of actions and percepts in it.

[28] In order to be able to implement software using such architectures, more detail and design must be considered.
[29] BDI architectures resemble human decision-making: belief is what the agent knows (its experience), desire is what the agent is willing to do, and intentions are desires plus the commitment to achieve them. In these architectures agents spent time not only doing things but also deliberating about what they should do. After choosing a plan to execute according to their current intentions, they are able to constantly introspect about the validity of such plan at each step of it, and if necessary, change the plan in order to better accomplish their goals.

3.4 Definition of Multi-Agent System

After introducing what an agent is and considering agents that operate alone, the agency theory can be extended to let agents interact within a so-called Multi-Agent System (MAS). According to [Wooldridge, 2002] "A Multi-Agent System consists of a number of agents which interact with one-another. […] To successfully interact, they will require the ability to cooperate, coordinate, and negotiate with each other."

In MAS there is no central control and all information and control is distributed among the various agents. **Figure A.5** presents an illustration of a MAS (based on [Wooldridge, 2002] and [Ferber, 1999]).

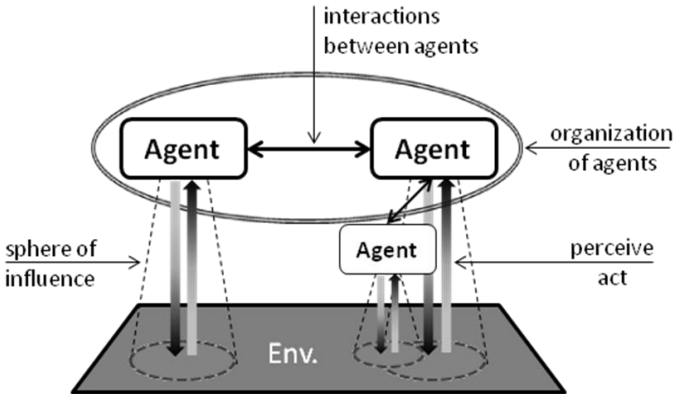


Figure A.5: A Multi-Agent System

As previously discussed, agents are able to perceive part (or all) of the environment and are also able to act upon part (or all) of it, depending on the type of environment (accessible or inaccessible). This can be represented by spheres of influence that determine the portion of the environment that one agent is able to interact with. Interactions between agents allow agents to cooperate, coordinate and negotiate as needed. Finally, interacting agents can also be organized to form higher-level organizational units.

To these concepts [Ferber, 1995] adds the concept of object as being a passive entity subject to manipulation (consumption and production) by agents (which in turn are defined as a subset of those objects, but with the ability to be active). All of these are also defined to be located, meaning that at each time it is possible to locate each object and agent within the environment. Finally Ferber introduces the idea that each agent possesses its own (internal) representation of the environment, of the objects and of other agents.

In this thesis the term Agent-Based Modeling will be used indistinctively from the term Multi-Agent System since the first simply refers to the process of modeling a certain reality using agents in order to build a multi-agent system[30].

[Wooldridge, 2002] defines the problem of how to build agents capable of independent, autonomous action so that they can successfully fulfill their objectives as *agent design,* while the issue of how to build agents capable of interacting with other agents to fulfill those objectives, especially when other agents cannot be assumed to share these same objectives (which is introduced by having multiple agents) as *society design*.

A concept that enables further elaboration on the agency theory in a multi-agent environment is the one of a *role*. A role is the functional or social part which an agent, embedded in a multi-agent environment, plays in a process like problem solving, planning or learning [Lind, 2008]. Roles are a useful abstraction considering the widely accepted meaning of the term in the real world that can be used to help describe and understand the system by describing the consistency of an agent's behavior within that system (as well as within the organization to which the agent belongs). Some roles present mutual dependencies and can only exist if other roles also exist (like the role of a teacher), and an agent can play several roles even at the same time (called role multiplicity). As in real societies, a role must be determined by a set of coherent behaviors, and not every set of behaviors constitutes a role.

[30] This is also supported by [Bousquet and Le Page, 2004]: "Recently, several researchers have started to use multi-agent systems, also called agent-based modeling, in different fields."

Using the more formal definitions already given, a role can be viewed as an extension of an agent's current knowledge (**D**), possible actions (**A**) and consequently it's *perceive*, *update*, *select* and *act* functions: $\mathbf{D}_{new} = \mathbf{D} \cup \mathbf{D}_r$ and $\mathbf{A}_{new} = \mathbf{A} \cup \mathbf{A}_r$ where $\mathbf{D}_r$ and $\mathbf{A}_r$ represent the knowledge and actions (respectively) of the added role **r**.

In MAS agents interact with each other, and this interaction can lead to cooperation (when different agents share objectives) or negotiation (when the interests are not aligned). Negotiation is the process of reaching agreements on matters of common interest [Wooldrigde, 2002]. Negotiation is very often needed in a MAS environment since each agent has its own goals which can be in conflict with other agent's goals. Alternatives to negotiation include simply doing nothing or trusting a third party (mediation). According to this author any negotiation setting has four components: a) a set of possible proposals that agents can make; b) a protocol; c) a (private) strategy for each agent to follow; and d) a rule that determines when a deal has been struck and what the agreement deal is.

Negotiation is not a simple task to accomplish within a MAS environment, since not only one-to-one negotiations can exist, but also between one agent and many others.

3.5 Origins of MAS in Relation to Other Disciplines

The origins of MAS can be found in several disciplines around the second half of the 20th century. The most cited disciplines involved are artificial intelligence, artificial life and computer science (although at its beginning very related to mathematics), but also with contributions from other fields such as economics and ecology.

The following figure exemplifies some of the milestones (and their dates) that led to the origin of MAS:

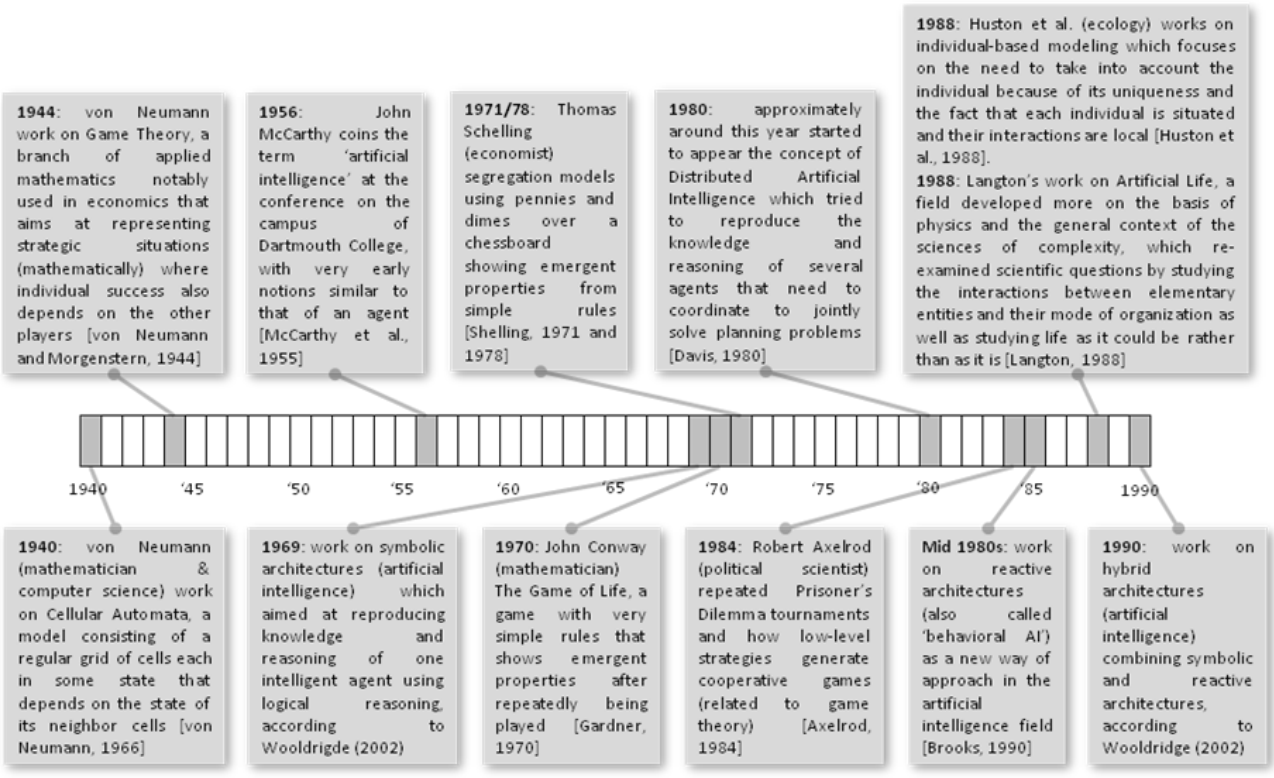


Figure A.6: Timeline of milestones that led to MAS

Following is a short list of the first applications of MAS to land-use systems (extracted from [Bousquet and Le Page, 2004]) since they relate to the topics tackled in this thesis:

- **1983**: Hogeweg & Hesper work on bee colonies [Hogeweg and Hesper, 1983]. Using a model that considered the interactions of individual bees, the authors show that the combination of the population dynamics of a bumble bee colony and simple behavior of the adult bees on the comb is sufficient to generate the social interaction structure of the colony.

- **1987**: Reynolds work on boids [Reynolds, 1987]. The author shows how the aggregate motion of an entire herd (of fishes or birds called 'boids') can be simulated by defining a set of simple rules over each one of them.
- **1994**: Lansing & Kremer work on water management (social-natural interactions) [Lansing and Kremer, 1994]. The authors show that Balinese water-temple networks dedicated to agricultural deities can have macroscopic effects on the topography of the landscape.
- **1994**: Bousquet et al. work on fisheries (social-natural interactions) [Bousquet et al., 1994]. The authors developed an object-oriented simulation model of the fishing in Niger as one of the first agent-based software models.

4. Agent-Based Modeling of Complex Systems

This section presents the applicability of the agent-based modeling approach to complex adaptive systems, and then discusses the suitability of the approach to agro-ecosystems as special case of complex systems.

4.1 Applicability of Agent-Based Models to Complex Adaptive Systems

Besides from the similarities between complex adaptive systems and the agent-based modeling approach that could already be noticed, this section further justifies the use of the latter to represent the former and gives a list of advantages of the ABM approach (including some comparisons with conventional approaches) for modeling complex adaptive systems.

The following four points help clarifying why the ABM approach is suitable for representing CAS:

- **Regarding Emergence**: ABMs allow to define the low-level behavior of each individual agent in order to let them interact (over time and space) to see whether some emergent property arises or not, and if it does, under which circumstances.
- **Regarding Self-Organization**: ABMs do not have any kind of central intelligence that governs all agents. On the contrary, the sole interaction among agents along with their feedbacks is what ultimately 'controls' the system. This lack of a centralized control is what enables (and enforces) its self-organization.
- **Coupled Human-Natural Systems**: ABMs allow considering together both, social organizations with their human decision-making and communications with biophysical processes and natural resources. This conjunction of subsystems enables ABMs to explore the interrelations between them, allowing analyzing the consequences of one over the other.
- **Spatially Explicit**: the feature of ABMs of being able to spatially represent an agent or a resource is of particular interest when communications and interactions among neighbors is a key issue. This can either imply some kind of internal representation of space or even the use of a Geographical Information System (GIS) with real data. This feature is of special interest in the case of agro-ecosystems.

Furthermore, [Miller and Page, 2007] introduce several benefits of a computational model such as ABMs over other more conventional approaches (such as mathematical or textual representations):

- **Flexibility vs. precision**: flexibility occurs when the model can capture a wide class of behaviors; while precision requires the elements of the model to be exactly defined. While long textual descriptions of elements and phenomena appear as a very flexible tool, natural language lacks precision. On the other hand, mathematical tools allow for precise descriptions (including formal problem-solving methods) with the cost of less flexibility. Computational models (such as ABM) present an interesting trade-off between flexibility and precision. These models allow flexibility while at the same time requiring precision in order to be able to implement it in the model (i.e. code it in a programming language);
- **Process oriented**: for a computational model to run, every aspect of how agents are allowed to interact must be well defined, and are the very basics of model behavior. Such details are often ignored in other approaches (like mathematical or economical models), and even when they could be incorporated, the inherent difficulties to do so often lead to gross simplifications or to rely on parameters to represent them;
- **Adaptive agents**: an important issue in social systems is how bounds on the ability of agents to rationally process information impact the behavior of that agent and of the whole system, and how does agent learning influences both. The flexibility of computational tools makes them

well suited for considering models of boundedly rational agents who adapt their behavior. Moreover, computational models of learning have been already developed in the field of computer science. Early works on this area focused on high-level cognitive (symbolic) systems, and while with initial success, real problems proved much more difficult than initially thought. Lower-level adaptive learning (like the already mentioned genetic algorithms) appears as an alternative to high-level cognitive systems.

- **System's dynamics**: agent-based models are inherently dynamic in the sense that the processes they model need not be in equilibrium. This allows studying systems in which equilibrium may or may not be reached. When transition paths between equilibrium states are short and conditions are stable, static models may accurately represent reality. In many systems though, this is not the case. Even when the conditions are right for equilibrium analysis, understanding the dynamics of the system may still be important. In situations where equilibria are a possibility, understanding the dynamics is likely to be insightful; in situations where equilibria are nonexistent or transition paths are long, understanding the dynamics is crucial.
- **Heterogeneous agents and asymmetry**: one of the key issues of ABM is that they allow for incorporating heterogeneous agents rather than having homogeneous agent behavior averaging out the differences. Whether a model should include heterogeneous behavior or not is an important issue and having an approach that allows for experimenting both scenarios is crucial. The amount of symmetry that the system presents and how much of that is actually modeled is also important. Symmetry assumptions usually simplify calculations, even though asymmetry may be an influential feature of the system.
- **Scalability**: the ability to solve a model analytically is often tied to the number of agents that are used. Thus, traditional methods typically focus on models composed of either very few or very many agents (e.g. competition models in economics or planetary motion models in physics). Once the behavior of a single agent is described, agent-based models can be easily scale up exploring the behavior of a system of arbitrary size by simply adding more agent instances. This scalability may be very important in many cases where just by adding one agent affects the whole system behavior (like in the case of monopolies or duopolies or in systems that present a tipping point[31]). Scaling also allows seeing emergent behavior that could not rise if the number of agents is not enough.
- Other advantages of agent-based computational models:
    - **Repeatable**: experiments can be repeated over and over again, and the same results can be obtained while maintaining the initial conditions. This enables extraordinarily precise manipulations in the virtual world that would be impossible in the real world.
    - **Constructive**: ABMs give a complete constructive and generative view of a phenomenon, from its origins to its manifestation. This ability to generate from the bottom up often can provide new insights and understanding.
    - **Low cost**: while developing the initial computational model can be costly, these models tend to be very cost effective, since the marginal cost of running and modifying is usually very low, allowing several runs to accommodate any statistical necessity.

The following Section presents how some special features of agro-ecosystems can be modeled using an ABM approach, which in conjunction with these previous advantages, make them the suitable tool for exploring such systems.

4.2 Modeling & Simulation of Agro-Ecosystems using an ABM Approach

Section 2.2 presented agro-ecosystems as a case of complex adaptive systems, while Section 4.1 showed the suitability of the ABM approach to model complex adaptive systems. Nevertheless, this section provides some views on the topic by different authors working on ABM approach to agro-ecosystems and presents some specific features of agro-ecosystems that can also be tackled with it.

According to [Ferber, 1999, pp 36]:

[31] A system is said to have a tipping point if by adding just one more element (e.g. an agent instance) the behavior of the entire system changes dramatically.

> *"Multi-agent systems bring a radically new solution to the very concept of modeling and simulation in environmental sciences, by offering the possibility of directly representing individuals, their behavior and their interactions […] it is thus possible to represent a phenomenon as the fruit of the interactions of an assembly of agents with their own operational autonomy."*

The environmental sciences had, according to [Bousquet and Le Page, 2004], for ten years the challenge to develop a new approach focusing more on the interactions between ecological and social components and taking into account the heterogeneity of these, where ecology, for which the environment is a fundamental notion, plays a key role in specifying concepts and developing appropriate tools.

The relations and heterogeneity in the social part of the MAS depend on the agent's social **neighborhood**, whereas the variability from the physical and ecological point of view can be achieved by considering spatial heterogeneity of the **environment** in the MAS. The neighborhood and the environment can be represented in different ways. For example, a tightly coupled representation is supported by cellular automata, and a rather loose coupling can be reached by implementing the social neighborhood as a network and the environment in a Geographical Information System (GIS).

A cellular automata model works with a grid of neighboring cells. Each cell represents a spatial unit and an individual or agent. Within the grid, the neighborhood is fixed (e.g. 4 or 8 neighboring cells if a square-shape cell grid is used), and more advanced models allow the use of non-local neighbors. Each cell can be at one of several states, and future states depend on transition rules based on a local spatiotemporal neighborhood [Parker et al., 2003]. Time advances in discrete steps. Cellular automaton are also very suited for using GIS since actual images or representations of real landscapes can be mapped to each cell, enabling more realistic analysis of the grid.

To address more complex neighborhoods and environments to simulate socio-ecological dynamics in agro-ecosystems and human decision making therein, more loosely coupled social and environmental representation in ABM are needed. The link between the two subsystems can be achieved by assigning a number of elements in the GIS (e.g. fields) to the responsibility of selected agents (e.g. farmers) as their sphere of influence. These kinds of joint work have led to many new applications of MAS for simulating land use and land cover changes, also known as MAS/LUCC [Parker et al., 2003].

The addition of spatially explicit features is especially important in agro-ecosystems where spatial heterogeneity can be relevant, for example by means of different land-uses, soil qualities, natural resources, etc. on each land-unit. Just as land-units can be organized to form higher-level spatial elements, social networks can be formed between agents or can emerge as a consequence of agent's behavior and interactions. What makes ABMs rather unique is that both of these organizational dimensions can be considered together, allowing exploring the consequences of one over the other.

Finally, it is worth noting that even though agents, from the ABM point of view, can be any autonomous and goal-driven entity, from the agro-ecosystems perspective agents are generally aspects of human societies or animal populations that organize among themselves, interact with the environment, and they are affected in their decisions by that environment [Matthews, 2006]. **Figure A.7** shows these relationships similarly as **Figure A.1** (in Section 1) showed an abstract complex system:

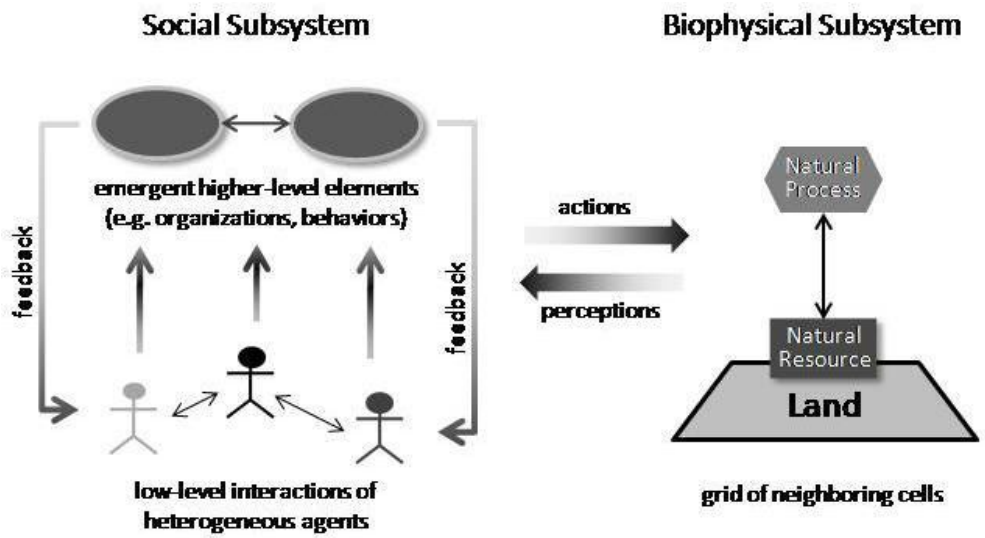


Figure A.7: Model of an Agro-Ecosystem using an ABM Approach

The actual implementation of an agent-based model is ultimately about developing a computational model, that is, software. Therefore, in order to analyze, design, implement and every other step involved in developing such systems, certain software development methodology must be used.

5. Software Development

5.1 Object Oriented Programming (OOP)

Most software is currently being developed using the object-oriented paradigm. Born with the ideas of the Simula programming language in 1965-67 [Birtwistle, 1973] and immediately followed by Smalltalk language around the 70's [Kay, 1993] object-oriented programming (OOP from now on, term coined by Alan Kay) becomes mainstream in the mid 1980's with the C++ programming language [Stroustrup, 1994] and even more widely used thanks to Sun's Java in the mid 1990's [Gosling, 2005] and from the beginning of 2000 with Microsoft's .NET Framework and the C# programming language [Microsoft, 2008].

OOP is a technique for programming, a paradigm, while an OOP language is a programming language that provides mechanisms that support the object-oriented style of programming [Stroustrup, 1991]. OOP also involves such concepts as encapsulation, modularity, data-hiding, message-passing, inheritance and polymorphism. The main idea behind this paradigm is to design certain object structure along with its behavior that allows objects to interact (by means of messages) to achieve some goal. The concept of object itself comprises internal data and operations that manipulate that data in an indivisible entity.

This is represented in **Figure A.8**:

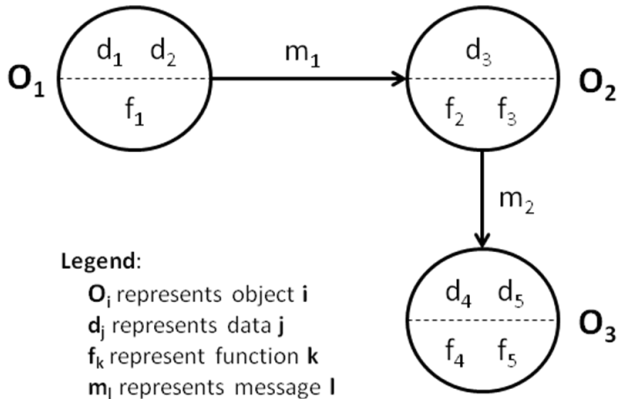


Figure A.8: Objects (data + functions) communicating via messages in OOP.

One of the most powerful features of OOP is that it serves as a metaphor[32,33] not only for implementing (coding) programs but also to design them (including design patterns [Gamma et al, 1994]) and to model reality in terms of objects and interactions (OO-analysis, including analysis patterns [Fowler, 1996]). All of this provides powerful abstractions that help bridging the gap between the (real-world) problem and the (software) solution, passing through OO-analysis and OO-design stages.

An example of such an abstraction that shows the power of this metaphor is the concept of object itself, which can represent anything in real life that has certain state and certain behavior. Each person has a name, age, hair color, etc. and each one of us can buy, sell, pay, etc. The state of each person will be represented by the attributes of the Person class and the behavior by the methods of the Person class. Another example is the possibility to define categories of concepts by using the OOP feature of inheritance.

5.2 Software development methodologies and tools

This thesis assumes that *software development methodology* is the same as *software development process* (or *software process* for short). This means to follow certain *process* in order to obtain a software product as the result of it[34]. [Pfleeger and Atlee, 2006] define a process in the context of software development as:

> "*A process is a series of steps involving activities, constraints, and resources that produce an intended output of some kind. A process prescribes all of the major process activities; […] uses resources subject to constraints; […] produces intermediate and final products; […] may be composed of subprocesses; […] each process activity has entry and exit criteria so that we know when the activity begins and ends; […] activities are organized in a sequence […]; has a set of guiding principles; […] and constraints and controls may apply to an activity, resource, or product.*"

A tool is an instrument or automated system for accomplishing something in a better way and a paradigm represents a particular approach or philosophy for building software [Pfleeger and Atlee, 2006].

According to [Booch, 1995] a software development process has four roles:

1) To provide guidance as to the order of a team's activities;

2) To specify which artifacts should be developed and when should be developed;

3) To direct the tasks of individual developers and the team as a whole; and

4) To offer criteria for monitoring and measuring the project's products and activities.

This thesis will focus on the first two of these objectives since it will focus on the specific activities and products that should be done rather than on the team involved or the process measures needed.

What lays in the core of all current software development methodologies is the *iterative and incremental* development of software. Traditionally software was developed using the so-called *waterfall* process [Royce, 1970], consisting of sequential phases of design, implementation and testing. In this sequential process, one phase must be ended before starting the next one (breaking the whole project based on these phases). In contrast, iterative and incremental processes iterate several times through a set of phases obtaining in each cycle a (partial) version of the product, each time bigger, until the final product is achieved by the succession of several iterations (breaking the whole project by subsets of functionality that are tackled on each iteration) [Pfleeger and Atlee, 2006; Fowler, 2003].

Figure A.9 shows a representation of an iterative and incremental process:

[32] "An easily overlooked benefit to the use of object-oriented techniques is the power of metaphor." [Budd, 2001]
[33] "OOP is an interesting example of a programming methodology explicitly organized around a powerful metaphor." [Travers, 1996]
[34] "A software process is a set of activities and associated results that produce a software product." [Sommerville, 2006]

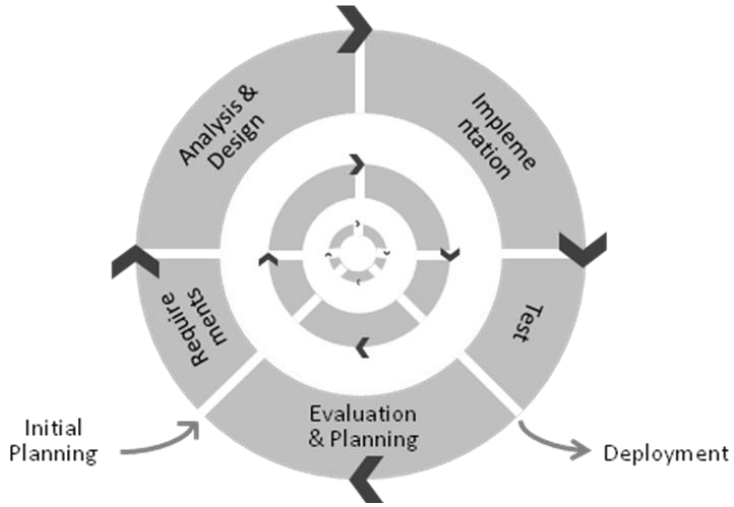


Figure A.9: Iterative & Incremental Process

According to Grady Booch[35] in [Kruchten, 2003] there are at least six software best practices that should be taken into account by the software process in order to successfully develop software. These are:

1) To develop software iteratively;

2) To manage requirements;

3) To use component-based architectures;

4) To visually model software;

5) To continuously verify software quality; and

6) To control changes to software.

These best practices are adopted in current mainstream software processes that are special cases of the iterative and incremental approach. These are Rational Unified Process (also known as RUP) [Kruchten, 2003] and eXtreme Programming (also known as XP, one of the most well-known agile processes[36]) [Beck, 1999]. Others include Microsoft Solution Framework (also known as MSF) [Turner, 2006] and Dynamic Systems Development Method (also known as DSDM) [Coleman and Verbruggen, 1998].

Each of these vary on the product deliverables (how many deliverables and when) as well as on administrative tasks (documentation, metrics, team work, etc.) but they all coincide in the use of iterations, mainly through the phases of analysis, design and implementation.

In relation to the fourth best-practice suggested by Booch (visually model software) the Unified Modeling Language (UML from now on) [Rumbaugh et al, 1998] is by far the most used graphical tool for modeling software. The UML has become a *de-facto* standard not only in industry but also in academy, created around 1995-96 by three very renowned authors in computer science (Grady Booch, James Rumbaugh and Ivar Jacobson) and is under the responsibility of the Object Management Group [OMG, 2008].

UML allows the visualization, specification, documentation and discussion of the artifacts that compose a software system. It is itself a specification composed of 13 different types of diagrams (in its 2.0 version) that provide graphical notation for modeling several aspects of a software system. All of these types of diagrams can be classified in structural diagrams and behavioral diagrams. The former are used to model structural elements and relationships of several software aspects while the latter are used to model dynamic behavior of elements (where time and sequencing are relevant).

Also, UML helps in the second best practice (manage requirements) by providing especial artifacts known as Use Cases along with related diagrams, and for the third best practice (use component-based architectures) by providing several diagrams for designing, documenting and discussing the system's architecture, each giving a different and complementary view.

[35] Booch proposed one of the earliest iterative processes in [Booch, 1983].

[36] According to Martin Fowler [Fowler, 2003] one of the authors of the agile manifesto: http://agilemanifesto.org

Even though UML is a very useful conceptual tool, it was not intended for implementation (although research is being conducted to allow UML diagrams to actually be executed[37]). Therefore when facing the implementation of a software design, other tools must be used, such as Integrated Development Environments (IDE).

6. Comparison Between Agent-Oriented and Object-Oriented Paradigms

In 1989 Yoav Shoham coined the term *agent-oriented programming* [Shoham, 2010] and defined it as a new programming paradigm based on a societal view of computation. This societal view implies (according to Shoham) ascribing mental qualities to agents such as beliefs, capabilities, choices and commitments (in the same way than BDI architectures propose to conceive agents in terms of beliefs, desires and intentions).

The idea behind these mental qualities or properties shared among several artificial intelligence and MAS-related authors [Shoham, 2010; McCarthy et al, 1955] include the use of mental constructs to design computational systems. This is the main difference between AOP and OOP: while OOP uses abstractions based on objects which are generic *entities* that comprise *identity* (a property inherent to all objects that allows to differentiate them), *state* (attributes by which the object can be characterized) and *behavior* (operations that the object can be asked to do by means of messages), AOP considers abstractions based on *human societies* and *mental properties*.

Even though the approach of using mental and societal abstractions gives powerful metaphors, there exists the risk of overusing them: everything can be considered to have mental or societal properties.

It is then straightforward that it is absolutely necessary to consider the AOP approach only 'when useful', which generally implies that the ascriptions help in understanding, analyzing, designing or implementing the problem at hand (thinking of light switches as agents just makes things more complex than adopting a more mechanistic approach). Therefore it will be more useful to ascribe mental qualities when applied to entities whose structure or functioning is not as well-known as the examples given.

Another important difference between both approaches is the autonomy that agents have. While in OOP objects just reply to messages sent from other objects in a predefined way, in AOP agents are supposed to be proactive and initiate actions over themselves or over other agents.

But AOP is not just OOP + proactivity. AOP can be viewed as a specialization of OOP. Whereas OOP proposes viewing a computational system as made up of modules (objects) that are able to communicate with one another and that have individual ways of handling incoming messages, AOP specializes it by fixing the sate (now called mental state) of the objects (now called agents) to consist of precisely defined components called beliefs (including beliefs about the environment, about themselves and about others), capabilities, choices and other similar notions. A computation consists of these agent's informing, requesting, offering, accepting, rejecting, competing and assisting one another. Regarding the fundamental concepts of each paradigm, objects in OOP can be compared to instance agents in AOP, and correspondingly, classes can be compared to agent types.

Furthermore, roles in AOP can be compared to interfaces in OOP since they specify the (partial) behavior of an agent type in one case and of an object in the other. A single class can present as many behaviors as interfaces it implements, the same way a single agent type can present as many behaviors as roles it plays.

It is also noteworthy that in OOP in order to define the entire behavior of a class it is not enough to only define its interfaces. First, because classes may have additional behavior that was not specified by any interface. Second, because it may be some relationship between the interfaces a class implements and the behavior that that class presents.

Translated this to AOP, if an agent type plays two roles at the same time, it may not be enough just defining those roles, since the agent type may behave differently when playing these two roles together, and this difference in behavior must not be described in either role since it is not specific to them, but of the agent type, so it must be defined within the agent type itself.

The following table summarizes these differences:

[37] This is known as Model Driven Architecture (MDA) [Mellor et al, 2002].

	OOP	**AOP**
Basic unit	Object (instance of a Class)	Agent instance of an Agent Type
Constraints defining the state of the basic unit	None	Mental properties (beliefs, desires, etc.)
Autonomy	Not necessarily present	May be present
Types of messages (interaction)	Unconstrained	Speech acts (inform, request, etc.)
Concept of role	Not necessarily present (comparable to using interfaces)	May be present

Table A.1: Comparison between OOP and AOP paradigms.

This short list of differences illustrates that even though OOP and AOP have important similarities, these arise because AOP can be seen as a *specialization* of OOP, a new paradigm thought as constraining the extremely general-purpose of the OOP paradigm and aligning those constraints to mental and societal properties.

The value (and innovation) of AOP is in providing useful abstractions for understanding and developing systems in terms of agents and societies of agents. In words of [Jennings, 2001]:

> "*When designing software, the most powerful abstractions are those that minimize the semantic gap between the units of analysis that are intuitively used to conceptualize the problem, and the constructs present in the solution paradigm.*"

Appendix B: Comparison between AOSE and OOSE Methodologies

The aim of this section is not to present an exhaustive comparison between the 16 general steps of AOSE methodologies and the steps and activities found in mainstream object-oriented methodologies, but to summarize the main differences and common points between them. Such a comparison can help in understanding the main ideas behind AOSE methodologies by contrasting them with much well-known OOSE methodologies.

The comparison will be based on general steps aiming at showing each methodology's suggestions about *what* to do rather than *when* to do it. In other words, the focus will be in comparing which steps are suggested in each methodology, in which order and which is the expected result. Concepts more specific to the life-cycle of each methodology (like for example iterations) and that help defining *when* the steps will occur will be left aside, not because they are not important but because the AOSE methodologies do not explicitly address them (maybe because they are not yet mature enough) and comparing their steps is considered to be of utmost importance. *How* to develop each step (for example what artifacts to deliver or what language to use) will be compared when possible.

Analogous to the 16 AOSE general steps, **Table B.1** presents such general steps but for mainstream Iterative & Incremental Methodologies[38]. For the sake of the comparison between both methodologies (AOSE and OOSE) the following table assumes that the system will be developed following some kind of component-based architecture. This allows to explicitly include in the table those steps especially suited for component development in the OOSE methodology, since they can be closely related to the concept of agent class in the AOSE methodology, allowing for a more precise (and real) comparison between them. This doesn't bias or restricts the OOSE methodology in any way since many mainstream of them explicitly call for using such architectures.

[38] Steps involved in Business Modeling are left out in order to concentrate on developing software and not on understating the business or organization in which the software will perform.

	OOSE Step	Description
System requirements and analysis steps	**1.** Identify overall system requirements	Involves eliciting both, functional and non-functional requirements (quality attributes) as well as identifying actors.
	2. Domain model conceptualization	Traditional conceptual diagrams the same as described for AOSE (typically using UML Class Diagrams as notation).
	3. Specify overall expected system behavior	Involves functional requirements analysis, typically with techniques such as Use Cases. Even though the behavior of the entire system should be specified, each component can be associated to a subset of these in order to maintain traceability and to better organize each component's development.
System architecture steps	**4.** Specify system architecture	Specify the system's overall architecture using components (component-based) like for example client/server, layered, etc.
	5. Identify component interfaces	Identify (at least in a broad sense) the interfaces that will be provided and consumed by each component. If possible, specify the behavior of each of these interfaces (operations).
	6. Specify component internal architecture	Specify each component's internal architecture, which will be encapsulated (and hidden) within that component.
System design steps	**7.** Specify component's object interactions	Specify, only for those requirements that should be fulfilled by each component, how the component's elements (classes, interfaces, objects, datatypes) will interact.
	8. Specify component's class structure	Based upon the previous step, specify the entire class structure for each component. This structure should populate the already defined component architecture (Step 6).
Implementation	**9.** Implement components	Implement (in an OO programming language) the detailed design made for each component in Steps 7 and 8.
Deployment	**10.** Specify component's deploy	Specify how many instances of each component will be deployed and over which node (processing and data-storage unit). This should be related (or it could be even already defined) to the overall system architecture (depending on how exhaustive the latter was).

Table B.1: Most common OOSE steps.

On the other hand, the 16 general AOSE steps can be found in Table 3 of Chapter 3.

Not surprisingly, AOSE methodology focuses of identifying agent classes (Step 3), specifying their interactions with other agent classes (Steps 5 to 7), their architecture (Step 8) and their interfaces (Step 10). This great deal of attention to agent classes' analysis and design derives from the crucial concept of agent autonomy, and even of agent mental attitudes (Step 9).

On the other hand, OOSE methodology focuses on the system as a whole, and classes (and their interactions) are only means to achieve the desired system-level functionality (in words of OOP, object collaborate with each other in order to achieve system requirements). Here, classes are designed altogether and organized with these system requirements in mind; while in AOSE each agent class has its own goals, which may or not mean collaboration, so their design must be much more independent from one another than in the OOSE case.

Actually, under the OOSE methodology, agent classes could be more appropriately compared with stand-alone components, since OOSE applies in such architectures the concepts seen in AOSE: interactions between components, individual component architecture, individual component internal design, component interfaces, communications between components, etc.

Nevertheless these differences, as Figure 7 of Chapter 3 showed, nearly all AOSE methodologies directly or indirectly derive and use the Object-Oriented paradigm, therefore huge differences should not be expected.

Appendix C: Case Study Glossary

This appendix presents basic definitions for several of the agronomical concepts used in Chapter 6 of this thesis.

Activity (or Production Activity)	Is any activity or management practice that a farmer/producer can perform, exploiting in some way its farm in order to obtain some benefit. Some production activities present in North Uruguay are cattle, soybean, or forestry.
Basalt	A volcanic rock. Basalt-soils are named because underneath the soil, basalt rock is present, sometimes having only few centimeters of soil above the basalt rock.
Born Calf	A newly born calf that has to spend some time lactating with its mother.
Breeder (or Cattle Breeder)	A farmer whose main production activity is livestock and specializes in
Calf	The young of domestic cattle.
Cattle	See Livestock.
Cohort	A group of people (in this case animals) who share a characteristic, usually age.
Condition Score (CS)	Also known as Body Condition Score: is a system for evaluating an animal level of body condition (amount of stored fat) and assessing a numeric score to facilitate comparison. In the context of this thesis, a cow with a CS of 2 would be dead or near death, while a CS of 3 would be minimal, and anything above will be considered better.
Ear Tag	A plastic or metal object used for identification of domestic livestock and other animals.
Empty Cow	A cow that is not pregnant.
Fertility Rate	In the context of this thesis, it represents the probability of a cow to effectively deliver after being mated. A percentage scale was used.
Flock	A herd of sheep.
Graze (or Grazing)	To put an animal to feed from natural grass. In the context of this thesis, this implies renting to other farmers their plots.
Heifer	A young cow before she has her first calf.
Herd	A group of animals.
Livestock	Also known as Cattle: domesticated animals raised in an agricultural setting to produce commodities such as food or leather.
Mating	The pairing of opposite-sex animals for copulation.
Plot	In the context of this thesis: the land plot that each farmer owns or rents for grazing.
Proactive Producer	In the context of this thesis: a producer more concerned with keeping reasonable condition scores in animals. This producer looks mainly at the grass availability, therefore anticipating the future animal's condition score.

Reactive Producer In the context of this thesis: a producer more concerned with keeping animals. This producer looks mainly at the animals and makes more radical decisions when they start to die.

Steer A young bull.

Stocking Rate A number that represents how many animal units a producer has. The bigger the stocking rate, the more animals the producer has. Each kind of animal (sheep and cattle) has different animal units: an adult cow represents 1.0 units in the stocking rate while one sheep 0.2 units. Each producer adjusts his stocking rate at different levels and at different seasons (e.g.: proactive producers adjust to 0.7 at winter while reactive to 1.2 in that same season).

Supplement A nutrient supplement given to cattle, generally made from grain, grass and fiber. Since it has certain cost associated, producers only invest on supplement when they feel it is absolutely necessary.

Wean Take the newly born calf away from his mother. It can be done naturally (after approximately 9 months) or artificially by the producer. In the latter, the producer must give supplement to the born calf in order to substitute his mother.

Appendix D: ICAART 2011 Paper

TOWARDS AN AGENT-BASED METHODOLOGY FOR DEVELOPING AGRO-ECOSYSTEM SOFTWARE SIMULATIONS

Jorge Corral and Daniel Calegari
Instituto de Computación, Universidad de la República, J.Herrera y Reissig 565, Montevideo, Uruguay
corral@fing.edu.uy, dcalegar@ fing.edu.uy

Keywords: Multi-Agent Systems, Agro-Ecosystem, Software Development, Simulation.

Abstract: The agent-based modeling (ABM) approach allows modeling complex systems, involving different kinds of interacting autonomous agents with heterogeneous behavior. Agro-ecosystems (ecological systems subject to human interaction) are a kind of complex system whose simulation is of interest to several disciplines (e.g. agronomy, ecology or sociology). In this context, the ABM approach appears as a suitable tool for modeling agro-ecosystems, along with a corresponding agent-oriented software engineering (AOSE) methodology for the construction of the simulation. Nevertheless, existing AOSE methodologies are general-purpose, have not yet accomplished widespread use, and clear examples of applications to agro-ecosystems are hard to find. This article sets the ground for a new software development methodology for developing agro-ecosystem simulations based on the ABM approach as well as on these already existing AOSE methodologies, but tailored to tackle specific agro-ecosystem features.

1 INTRODUCTION

Many of the current challenges and opportunities (e.g. globalization, sustainability, terrorism, epidemics or climate change) can be seen as complex systems (Miller and Page, 2007). Understanding the components, behavior and interactions in these systems is the first step to whatever analysis is needed about them. Models, as simplifications of certain reality or problem, are a fundamental tool to this aim. Moreover, the possibility of direct experimentation over these systems is rare if not impossible, so the need for simulation becomes imperative. Even though not at global scale, agro-ecosystems are complex systems, were modeling and simulation allow for understanding system dynamics as well as to explore future scenarios.

Several approaches or methodologies can be used for modeling an agro-ecosystem. In particular, the agent-based modeling (ABM) approach appears as a suitable tool for its aim (Miller and Page, 2007). In the other hand, simulating generally means developing a software system, representing the agent-based model. This requires the use of some agent-oriented software engineering (AOSE) methodology.

Several AOSE methodologies are currently available for guiding a programmer in order to develop software following the ABM approach (Henderson-Sellers and Giorgini, 2005). However, they are general-purpose methodologies, so the programmer is faced with a trade-off between using an already existing one and not leveraging the specificities of agro-ecosystems, or to follow an ad-hoc methodology that pays detail to those features. Up to our knowledge there are no specific AOSE methodologies for this purpose. Nevertheless, there are some related work (Le Page and Bommel, 2005) that address the simulation of agro-ecosystems using an ABM approach without a methodological framework.

In this work we propose an AOSE methodological framework for modeling and simulating agro-ecosystems that is based upon other general-purpose AOSE methodologies, relies on a standard graphical modeling language (UML), covers the most common agro-ecosystem features, and is intended for easy uptake from programmers only knowing object-oriented programming. The methodological framework is not a methodology

itself. It focuses on identifying relatively general steps and artifacts (produced by the steps) that could be ensemble for developing such a simulation, whilst other methodological aspects, as defining a strict order among steps and identifying roles for that steps, are not yet considered.

The rest of the paper is structured as follows. Section 2 briefly presents the background of our work. Section 3 resumes existing AOSE methodologies, while Section 4 describes how these methodologies are considered in a methodological framework for the development of agro-ecosystem simulations. Section 5 describes how agro-ecosystem features are supported in the proposed methodological framework, and Section 6 presents a case study which stands as a proof of concepts. Finally, the conclusions and an outline for further work are described in Section 7.

2 Background

This section first introduces complex systems and presents agro-ecosystems as a special case of these. The ABM approach is then described and applied to represent an agro-ecosystem.

2.1 Complex Systems

Complex systems are composed of a large number of interacting elements, and two properties set a complex system apart from one that is merely complicated: emergence and self-organization. Emergence is the appearance of behavior that could not be anticipated from the knowledge of the parts of the system alone. Self-organization means that there is no external controller or planner engineering the appearance of the emergent features; they appear spontaneously (CSIRO, 2008).

A key feature of real systems that has proved to be essential in the appearance of rich emergent features is local interaction. In other words, elements of a system only interact with their neighbors (CSIRO, 2008). Well-known and simple rules in the micro-level can make the emergence of system-level phenomena.

Emergent properties and complexity usually arise when the relations among the elements of the system are not linear. This means that the behavior of a single element is not the result of a linear combination of the individual behaviors of related elements, leading to a non-linearity. Another source of complexity appears when there are time and scale differences between cause and effect. That is, if an action in certain level of organization or hierarchy at a certain time ends up having its effect over a long period of time and over different levels of organization or hierarchies (different scales), then the understanding of the cause→effect relationship vanishes.

Complex Adaptive Systems are defined as systems that are capable to adapt and self-organize in response to perturbations or distortions in the environment or by the result of certain interrelations between the elements. System adaptation is ultimately concerned with the adaptation of each individual element of the system, since there is no centralized control and therefore no single element that represents the entire system. This also relates to the concept of co-evolution of the different elements or parts of the system, which by means of their interrelations evolve their behavior over time, adapting themselves to new situations (Rammel, Stagl and Wilfing, 2007).

2.2 Agro-Ecosystems as a Case of Complex Systems

An agro-ecosystem is the human manipulation and alteration of ecosystems for the purpose of establishing agricultural production (Gliessman, 1997). Agro-ecosystems results from the interplay between endogenous biological and environmental features of the agricultural fields and exogenous social and economic factors, and is delimited by arbitrarily chosen boundaries.

Agricultural strategies respond not only to environmental, biotic, and cultivar constraints, but also reflect human subsistence strategies and economic conditions (Ellen, 1982). This stresses the importance of social factors like labor availability, access and conditions of credit, subsidies, perceived risk, price information, association obligations, family size, and access to other forms of livelihood are often critical to understanding the logic of a farming system (Hecht, 1987).

Any system uses its processes and resources to convert inputs into outputs. Concerning the resources commonly found in agro-ecosystems, Norman (1979) suggests the following classification: **1) Natural resources**: the given elements of land, water, climate and natural vegetation that are exploited by the farmer; **2) Human resources**: the people who live and work within the farm and use its resources for agricultural production; **3) Capital resources**: the goods and services created, purchased, or borrowed by the people associated with the farm to facilitate their exploitation of natural resources for agricultural production; and **4) Production resources**: the agricultural output of the farm such as crops and livestock.

Any ecosystem by itself can already be considered as a case of complex adaptive system, considering its various components, organization levels, micro and macro interactions and their feedbacks, and even more if including a social subsystem (with heterogeneous behaviors) as in the case of agro-ecosystems. The interactions between

the social and natural subsystems inside an agro-ecosystem, such as farmers' practices affecting natural resources, are examples of local interactions at a small scale which may produce negative or positive feedbacks affecting in turn the decision-making of social actors. Repeating these interactions on a daily basis can produce emergent properties and new organizations across the agro-ecosystem in the long-term.

According to Dawn, Hessl and Davis (2008) the sources of complexity in coupled human-natural systems (as is the case of agro-ecosystems) arises mainly from temporal, spatial and scale mismatches between actions and their impacts which occur because of the indirect and imperfect nature of these interactions between human actions and their impact on the environment. The sources of adaptation come from the capacity of humans to adapt their behavior by learning from their experience, and from the capacity of this environment to adapt to new conditions and constraints. According to Perez and Batten (2006) these kinds of coupled systems, involving people, other living entities, an environment, information exchange and the co-evolution of all of these things over time, are inherently complex and adaptive due to the ability of human beings to switch from rational to deductive reasoning.

2.3 Agent-Based Modeling

There is no consensus on the definition of agents. A well-known definition that supports the computer science focus presented in this article is proposed by Wooldridge (2008): "*An agent is a computer system that is capable of independent action on behalf of its user or owner, figuring out what needs to be done to satisfy design objectives, rather than constantly being told [what to do]*". In this definition, the word 'independent' refers to agent autonomy, capable of acting independently and exhibiting control over its internal state. Thus an agent is a computer system capable of autonomous action in some environment in order to meet its design objectives (Wooldrige, 2002 and 2008).

In order to let agents react (or take the initiative) according to changes in the environment, agents must perceive their environment and have some way to act upon it, after reasoning what to do. This leads agent to a Perceive/Reason/Act cycle, shown in Figure 1.

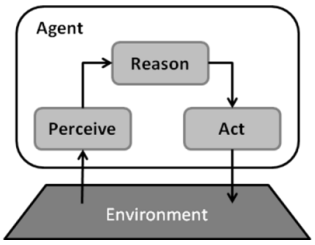


Figure 1: General representation of an agent with its environment.

The agency theory can be extended to let agents interact within a so-called multi-agent system (MAS). According to Wooldridge (2002) "*A Multi-Agent System consists of a number of agents which interact with one-another. […] To successfully interact, they will require the ability to cooperate, coordinate, and negotiate with each other.*" In MAS there is no central control and all information and control is distributed among the various agents. Figure 2 presents an illustration of a MAS based on (Wooldridge, 2002) and (Ferber, 1999).

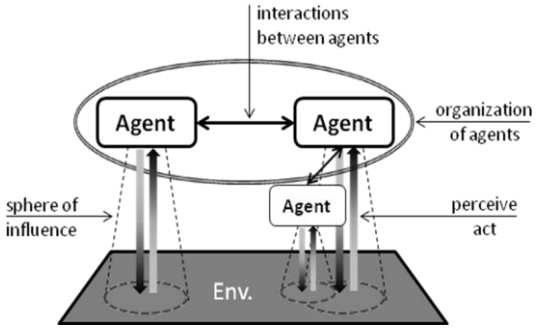


Figure 2: A Multi-Agent System.

As previously discussed, agents are able to perceive part (or all) of the environment and are also able to act upon part (or all) of it depending on the type of environment. This can be represented by spheres of influence that determine the portion of the environment that one agent is able to interact with. Interactions between agents allow agents to cooperate, coordinate and negotiate as needed. Finally, interacting agents can also be organized to form higher-level organizational units.

A concept that enables further elaboration on the agency theory in a multi-agent environment is the one of a *role*. A role is the functional or social part which an agent, embedded in a multi-agent environment, plays in a process like problem solving, planning or learning (Lind, 2008).

Roles are a useful abstraction considering the widely accepted meaning of the term in real world that can be used to help describe and understand the system by describing the consistency of an agent's behavior within that system (as well as within the organization to which the agent belongs).

In summary, the ABM approach enables the simulation of heterogeneous populations of interacting agents, which can also contain non-agent passive objects (commonly referred to as resources). The agents can exhibit a set of different behaviors, and the mechanism of selection can range from simple procedural logic to highly sophisticated reasoning. The repertoire of behaviors can be fixed or extendible, and the latter implies that the agents should be able of learning. The intelligence of the agent depends on its abilities to reason and to learn. The adaptive behavioral patterns enable self-organization of the population and can result in emergent phenomena. Consequently, the ABM approach is suitable to address complex adaptive systems.

2.4 Agent-Based Modelling of Agro-Ecosystems

Since the ABM approach can be used to model complex adaptive systems, and since agro-ecosystems are a special case of these, it naturally follows that the ABM approach is suitable for modeling agro-ecosystems. There are also numerous arguments that support this, like those presented in (Miller and Page, 2007), but nevertheless it is worth analyzing how the ABM approach specifically addresses the following issues, which characterize an agro-ecosystem: **1) Emergence**: ABMs allow to define the low-level behavior of each individual agent in order to let them interact to see whether some emergent property arises or not, and if it does, under which circumstances; **2) Self-Organization**: ABMs do not have any kind of central intelligence that governs all agents. On the contrary, the sole interaction among agents along with their feedbacks is what ultimately 'controls' the system; **3) Human-Natural Systems**: ABMs allow to consider together both, social organizations with their human decision-making and social communications and biophysical processes and natural resources. This conjunction of subsystems enables ABMs to explore the interrelations between them, allowing to analyze the consequences of one over the other; and **4) Spatially Explicit**: the feature of ABMs of being able to spatially represent an agent or a resource is of particular interest when communications and interactions among neighbors is a key issue. This feature is of special interest in the case of agro-ecosystems.

Resources are not generally defined as a separate aspect of an ABM. However, there are particularly useful in the context of agro-ecosystems, where they are generally thought of as passive objects that are, for example, produced by the environment and consumed by agents within certain process (like crop production), in contrast to agents which are (pro)active objects with decision-making capacities that allows them to affect those resources

The actual implementation of an agent-based model is ultimately about developing a computational model, that is, software. Therefore, in order to analyze, design, implement and every other step involved in developing such systems, certain software development methodology must be used.

Since these coupled human-natural systems cannot be manipulated and tested as other systems, due to scale and resource difficulties, the possibility of simulating them is crucial. The objective of simulating this kind of systems is to prospect scenarios were the interest is not in the 'fortune-teller' features of the simulation but on discovering possible outcomes under certain conditions, exploring the consequences of manipulating certain system. Another objective can be to deepen the understanding (and also possibly learning/teaching) of a coupled human-natural system, since the more micro/macro level study of each part as well as their interactions require an in-depth understanding of them in order to obtain a simulation that is close to reality.

In the next sections we address the problem of developing an agent-based methodology for the simulation of agro-ecosystems.

3 Existing AOSE Methodologies

Even though there are several methodologies for developing agent-oriented software systems, this article is based on those ten presented in Henderson-Sellers and Giorgini (2005) since they are arguably the most referenced and cited elsewhere. These are: GAIA, Tropos, MAS-Common CADS, Prometheus, Passi, Adelfe, Mase, Rap, Message, and Ingenias. From these ten AOSE methodologies, Tran and Low (2008) summarize 16 general steps, which are shown in Table 1.

At this point, it is reasonable to think that any new AOSE methodology will be strongly related with these 16 general AOSE steps. Nevertheless, notice that these steps are not tailored for *simulating* an agent-based model, neither for developing *agro-ecosystem simulations*. Up to our knowledge there are no methodologies for these purposes. These will be indeed the drivers of the methodological framework which is proposed in the next section.

Table 1: General AOSE steps taken from ten AOSE methodologies.

	AOSE Step	Description
Problem domain analysis steps	**17.** Identify system functionality	Determine what the system should do in terms of functionalities (the same as traditional requirements engineering techniques).
	18. Identify roles	Identify the roles that should later be played by the different agents. This step may include analyzing the organizational context in which the MAS will be deployed since roles commonly (and naturally) appear in real organizational contexts.
	19. Identify agent classes	Identify which types of agents are needed. This can be related to roles in the sense that a certain agent class can be created to cope with certain role.
	20. Model domain conceptualization	Identify the domain's main concepts and relationships.
Agent interaction design steps	**21.** Specify acquaintances between agent classes	Determine basic relationships between agent classes (such as who knows who, possible hierarchies, etc.)
	22. Define interaction protocols	Define how agents can interact by defining communication protocols between them.
	23. Define content of exchanged messages	Define what the messages between agents will contain. These 3 steps (which comprise the "Agent Interaction Design") are strongly related and can be considered altogether.
Agent internal design steps	**24.** Specify agent architecture	Determine the specific (internal) architecture for each agent class. The specific architecture to choose will depend on the characteristics that the different agent classes should present (e.g. if they must conceptualize beliefs, goals and plans, or if they will be just reactive).
	25. Define agent mental attitudes	If the agent is to present mental attitudes like goals, beliefs, plans or commitments, define these for each agent class. This step includes trying to identify how these attitudes will be design internally for the agents' classes that need them.
	26. Define agent behavioural interface	Determine the capabilities, services, contracts and any other interface that the agent must provide. This will depend on the agent architecture.
Overall system design steps	**27.** Specify system architecture	Overview of all components and their connections (in a higher level than Agent Architecture). This step also includes resources and environment (if they are of interest) and their relation to the agent classes.
	28. Specify organizational structure/inter-agent social relationships	If the system can be analyzed in terms of organizational concepts, then specify the organizational structure and the inter-agent social relationships. This step can be related to Step 2 (Identify Roles) since roles are generally crucial in any organizational structure, as well as hierarchies between them.
	29. Model MAS environment	If the environment is relevant, then identify its resources, facilities and characteristics. Although part of this step can be done in Step 12 (mainly those structural environmental aspects) others (mainly those functional aspects) may not.
	30. Specify agent-environment interaction mechanism	Closely related to the previous two step, involves detailing how the different agent classes are to be related to their environment (analogous to interaction protocols but instead of agent-agent interactions agent-environment interactions).
	31. Instantiate agents	Determines how many agent instances will be needed for each agent class.
	32. Specify agent instances deployment	Determine where the agent instances are to be (physically) deployed. This includes specifying agent platforms, nodes (with processing power), connections, etc.

4 Methodological Framework

The purpose of our work is to propose an AOSE methodological framework for modelling and simulating agro-ecosystems. We want this methodological framework to be based on the general AOSE steps introduced in the previous section in order to strengthen the proposal.

We also want to rely on a standard graphical modelling language in order to support the development process by facilitating the interchange between domain experts and simulation developers. In this article we use plain UML 2.0 (OMG, 2005). However, the methodology will be best supported by a more specific language for representing software systems based on software agent concepts, e.g. AML (Cervenka and Trencansky, 2007) and those diagrams promoted by the Foundation for Intelligent Physical Agents [FIPA, 2010]. Since these languages are not tailored for agro-ecosystems simulations, it must also be adapted to our methodological framework (in future work).

Finally, we will assume that the simulation will be finally developed within an existing simulation framework software package, e.g. CORMAS (CORMAS, 2010). However, we will not make any assumption about what simulation paradigm it uses, i.e. continuous, time-step or discrete-events. This will be indeed later matter of discussion.

We present next the methodological framework, by identifying relatively general steps and artifacts (produced by the steps) that could be ensemble for modelling and simulating agro-ecosystems.

4.1 Requirements

In order to model and simulate and agro-ecosystem using an ABM approach, the following elements should be taken into account: **1) Agents**: they will naturally be at the core of any attempt to develop a simulation using an ABM approach. This includes interactions between agents; **2) Environment**: is what surrounds the agent, where the agent is located, and a means by which the agent receives input; **3) Resources**: represent what the agents produce and/or consume; and **4) Simulation Capabilities**: since the objective is to simulate agents, environment and resources, the methodological framework must allow this to happen. This includes (but is not limited to): the notion of time (or time steps), the possibility to configure different initial situations from which to start the simulation, configuration parameters that go along with the simulation run, and output visualization.

4.2 Selecting Steps

Each one of the 16 general AOSE steps was analyzed in order to determine its convenience for the methodological framework, and even though a thorough study has been conducted for each step. Table 2 summarizes the results obtained. Most of the selected steps were renamed in order to better understand its purpose in the context of MAS simulations.

4.3 New Steps

The steps analyzed before do not specifically cover simulation capabilities. To this end, three more steps are added.

The ***7. Simulation Configuration*** step aims at defining those fundamental elements that will enable the simulation to be run, namely: **1) Initial Configuration**: determine the state of the simulation at time zero (e.g. initial number of instances for each agent type and their distribution over the environment, initial volume of resources and their location, etc.); **2) Time Definition**: determine how time passes, i.e. continuous, fixed steps or event-based (e.g. agent instances are called upon their behavior twice a year if the time step is defined to be six months); **3) Task Scheduling**: determine what to do while time passes: the order in which the behaviors of agents is called upon, resources evolve and environment changes (e.g. at each time step, first let agents execute and then let the resources evolve); **4) Input Parameters**: determine which data is needed as input to the simulation (e.g. the price of oil and its evolution); **5) Output**: determine the calculations and variables the simulation is to generate (e.g. producers' income) and how often this output is generated; and **6) Visualization**: determine which output parameters are to be visualized by the user and how (e.g. producers' income as a numerical value and land cover as colored cells in a grid).

The ***8. Implementation*** step involves coding the agent-based simulation using an already existing simulation framework software package. The developer should have enough elements (as well as enough understanding of the problem) to start coding. In this sense, the developer should be based on the information provided by the artifacts developed in the previous steps.

The ***9. Simulation Run & Sensitivity Analysis*** step involves gathering all the necessary real-world

data (including possible historical data) for allowing the simulation to be run and to perform explorations about how the output of the simulation is affected when certain elements are changed.

Table 2: Summary of the steps selected from the 16 general AOSE steps with their corresponding artifacts.

	General AOSE Step	Selected Step	Analysis
Problem domain analysis steps	**1.** Identify system functionality	1. Identify System Purpose	Since we are concerned with simulations, identifying the system functionality is transformed into identifying the system purpose (what should the system simulate and why), including those questions that the system should answer (the objective of the simulation).
	2. Identify roles	2. Identify Roles and Agent Types	The system will probably represent several different roles people play in an agro-ecosystem (e.g. farmer). Roles help in determine agent types (or 'agent classes' as called in the general step). Moreover, agent classes can be viewed as specializations of roles, more concerned on agents than in social behaviors (as roles do). Since these concepts may be crucial for agro-ecosystems, these steps will be considered together.
	3. Identify agent classes		
	4. Model domain conceptualization	3. Model Domain Conceptualization	It is very useful to have a structural overview of the agent types, the resources, and the most relevant elements that compose the environment, as well as its relations, so this step is selected as is.
Agent int. design steps	**5.** Specify acquaintances between agent classes	This can be either included when identifying agent types and roles (*2. Identify Roles and Agent Types*) or when modelling the domain conceptualization, by means of associations between concepts (*3. Model Domain Conceptualization*), so this step will not be considered by itself but as part of previous steps.	
	6. Define interaction protocols	4. Define Agent Interaction	In the case of developing an agro-ecosystem simulation, agent types will be developed altogether, so the autonomy and independence of individual agents is much reduced compared to other MAS scenarios. This facilitates the task of communication between agents, and enables considering these two steps as one.
	7. Define content of ex-changed messages		
Agent internal design steps	**8.** Specify agent architecture	5. Agent Architecture and Design	This step should generate enough information for detailing the agent type's internal design, describing how the agent will be able to follow the Perceive/Reason/Act cycle already presented.
	9. Define agent mental attitudes	If mental attitudes are to be considered for certain agent type then these attitudes will be defined in the previous step (*5. Agent Architecture and Design*).	
	10. Define agent behavioural interface	The agents of the agro-ecosystem will not provide services to other software entities (as agents in other contexts may do) so the "services" they have are actually the messages they will exchange between them (considered in *4. Define Agent Interaction*) and with the environment (considered in *6. Model Environment and Resources*).	
Overall system design steps	**11.** Specify system architecture	The overview of components and connections is already given by the context of agro-ecosystem simulations, and also by the architecture imposed by the simulation framework software package chosen for implementing the simulation over it (ej: CORMAS).	
	12. Specify org. structure/inter-agent social relationships	Even though agro-ecosystems may involve some kind of organizational structure they are not driven by any organizational metaphor as this step assumes. Any 'inter-agent social relationship' was already defined in "*2 Identify Roles and Agent Types"* and in "*4 Define Agent Interaction"*.	
	13. Model MAS environment	6. Model Environment and Resources	The environment is indeed a crucial element to consider. Since resources are within the environment, they are considered altogether. This step provides a detailed view of both, extending its structure (considered in *3. Model Domain Conceptualization*), and defining its behavior.
	14. Specify agent-env. interaction mechanism	In an agro-ecosystem simulation, the agent behavior strongly depends on the environment so this interaction mechanism could be either considered in "*2. Identify Roles and Agent Types*" and in "*4. Define Agent Interaction*".	
	15. Instantiate agent classes	This step is part of the initial configuration of the simulation as considered later.	

	16. Specify agent instances deployment	Since the entire agro-ecosystem simulation will be run in only one computer, there is no need to consider physical distribution.

4.4 Steps and Artifacts

Table 3 summarizes the steps of the proposed methodological framework, including its aim and artifacts. Since we do not have yet any specific modeling language tailored for agro-ecosystems simulations, for the sake of a better understanding we consider here plain UML 2.0 diagrams. Since agro-ecosystems are a kind of complex system (**Section 2.2**) and these can be modeled using the ABM approach (**Section 2.4**), it naturally follows that agro-ecosystem can be modeled and simulated using an ABM approach, and particularly following the methodological framework introduced in this section. Nevertheless, there are quite specific agro-ecosystem features that are worth analyzing how could they be represented, modeled and simulated by this methodological framework. This is analyzed in the next section.

Table 3: Summary of steps that compose the proposed methodology.

Proposed Step	Aim of the Step	Artifacts
1. Identify System Purpose	Define the purpose of the simulation, including its objective and questions to be answered.	Text Document including the purpose of the simulation, an overview of the context in which the simulation will be developed, including why the simulation will be developed and what will be expected from it.
2. Identify Roles and Agent Types	Identify agent types and roles, especially agent behavior.	Text Documents for role's identification and description and for identifying agent types and their relation to roles; UML Activity Diagrams for agent type's behavior specification.
3. Model Domain Conceptualization	Depict the structure of the problem, including entities and relationships.	UML Class Diagram for modeling main concepts, including those in steps 2 and 6.
4. Define Agent Interaction	Determine when, how and what the different agents will communicate.	UML Sequence Diagrams for modeling interactions.
5. Agent Architecture and Design	Define internal agent design (structure and behavior) in order to fulfill its perceive/reason/act cycle, within a simulation framework software package.	UML Class Diagram for designing the internal structure of each agent type, as well as any other UML Structure Diagram (like Package and Component Diagram); and UML Communication Diagrams for designing the internal behavior of each agent type, as well as other UML Behavioral Diagrams (like Activity and State-Transition Diagrams).
6. Model Environment and Resources	Determine behavioral aspects (evolution) of resources and environment, and completing structural aspects.	Text Document for complementing other diagrams; UML Structure Diagrams for further modeling structural aspects of the environment and resources; and UML Behavioral Diagrams for modeling functional (behavioral) aspects of the environment and resources.
7. Simulation Configuration	Define those fundamental elements that will enable a simulation to be run.	Text Documents for documenting the configuration; UML Object Diagram for the initial configuration; and UML Sequence Diagram for tasks scheduling.
8. Implementation	Codify the simulation.	Code.
9. Simulation Run & Sensitivity Anal.	Answer the simulation's objectives and questions.	Text Documents with the conclusions.

5 MAPPING AGRO-ECOSYSTEM features

This section presents those features that characterize an agro-ecosystem and shows how they are covered by the methodological framework proposed in the previous section. The objective is to show that this methodological framework can successfully represent them with the available steps and artifacts, leaving no relevant agro-ecosystem feature uncovered. The features considered here are schematically presented in Figure 3.

Three of the features are outside the limits of the agro-ecosystem: climate, government and market prices. Nevertheless they are relevant because they affect it in some way that is of interest, but since they do not belong to the agro-ecosystem they will not be explicitly modeled (i.e. there will not be artifacts representing them) but considered mainly as inputs to the agro-ecosystem.

Table 4: Summary of agro-ecosystem's features and their representations within the proposed methodology.

	Feature	Representation
Natural Resources	Land Plots	Plots can be thought of as land components, giving more flexibility by decomposing the land in several elements. About each plot, it can model its location (e.g. by coordinates), its vicinity (e.g. by associations with neighboring plots) as well as the relation between agents and plots (e.g. representing ownership) and between plots and resources (e.g. representing which resources are hosted in which plots). Plots allow for an easy geographical representation of land, where each plot is represented by a cell in a grid that can be shown to the user (visualization).
	Water	As a special kind of land plot or as a resource within a land plot.
	Climate	As an input in the 7. *Simulation Configuration* step.
	Vegetation	As an attribute of the land plot or a resource within a land plot.
Human Resources	Farmers' Types	As agent types. Since the behavior of a farmer can dynamically evolve over time, different agent roles can be defined.
	Organization of Farmers	As a new agent type that is related to its members (which are other agent types). If belonging to an organization implies certain behavior in its members, then new roles can be defined for them and agents should be able to change its behavior when playing this new role.
Capital Resources	Goods & Services	These are generally of two kinds: those which are used as inputs for production (like fertilizers and machinery) and those which are used as family consumption (like the electricity bill, health care or a new car). The former has to do with resources (and in some cases represented as an attribute of a farmer) while the latter with farmers' livelihood, needs and expectations (considered within the behavior of farmers).
	Money, Savings & Loans	Similarly to the last feature, the concepts here involved may be divided in two: the activities that led to an increase/decrease (e.g. buying or selling) and the quantities (e.g. current amount of money the farmer has or how much debt he/she owes). The former can be modeled in the behavior of the farmer, and the latter as an attribute of it.
Prod. Res.	Crops & Livestock	As resources which can either be associated to a certain land plot (as in the case of crops) or not (livestock). If a resource presents certain dynamics, these may be expressed, much like the behavior of agents, but applied to resources (which are passive in contrast to agents).
Other Features	Natural Processes	Since natural processes are not resources themselves, but are closely related to them, it is useful to conceive both at the same time within the environment. As the Crops & Livestock feature, resources may present certain dynamics to be run during the simulation, so it should be clearly defined whether these dynamics actually represent natural processes or if they are only concerned with the resource itself. This implies that natural processes may be explicitly modeled as another feature of the agro-ecosystem, or implicitly considered when modeling resources.
	Landscape	As an aggregation of all land plots. The initial landscape can be determined by the modeler in the initial configuration, e.g. by determining how many plots are used for agriculture.
	Market Prices and Evolution	Since it is generally the case that the price over which production resources are sold is not controlled nor determined by the system under study (e.g. a farm), prices are considered as an external input to the simulation, and therefore it is considered as an input parameter.

	Government Policies	This feature is not directly modeled into the simulation since the government is outside the boundaries of the system. The interest is to compare the evolution of the simulation with and without the introduction of certain government policy. This requires modifying the simulation in order to take them into account. The modifications can range between changing input parameter values (e.g. because of an increase in taxes) and changing the behavior of agents (e.g. by introducing new ways of associations of farmers). This changes lead to the exploration of different scenarios.

Finally, one feature partially appears as part of the agro-ecosystem, the landscape, since it is collectively constructed with other elements outside the boundaries of the agro-ecosystem.

In Table 4 we summarize how each of these agro-ecosystem features can be represented by the proposed methodological framework. The features were organized around the four categories of resources (in the sense of an agro-ecosystem, not in the sense of ABM) introduced in **Section 2.2** plus a fifth category including other features commonly in this context: natural processes, landscape, market prices and evolution, and government policies.

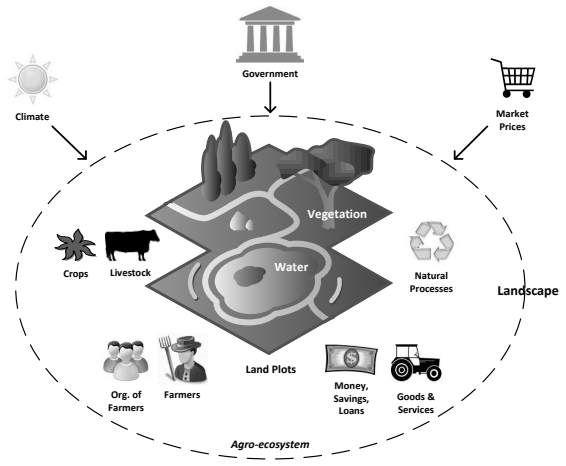


Figure 3: Schematic representation of 13 relevant agro-ecosystem features.

6 CASE STUDY

This section exemplifies the proposed methodological framework. The case study shows the dynamics between cattle and crop production in a certain basin and it is based in a real world situation (further details are omitted to keep it anonymous). For each step of the proposed methodological framework, an exemplifying artifact will be presented.

It is worth noting that even though the steps can be executed in an iterative and incremental way (as mainstream software development methodologies suggest), in order to simplify the exposition, the following steps are presented in a sequential order. Finally, all but the last two steps (*8. Implementation* and *9. Simulation Run & Sensitivity Analysis*) are exemplified with the Case Study since they do not reveal any singularity from the proposed methodological framework point of view.

Step 1: Identify System Purpose

The purpose of this simulation is to understand the long-term consequences on land use of producer's strategies, regarding cattle and soybean production in the basin.

The general context is that traditional producers of the basin historically had a production strategy based on looking for a balance between cattle and agricultural production, alternating pasture and crops. Recently (approx. in the year 2000) a new kind of producer entered the system: investment fund managers. These are multinational companies whose strategy is continuous soybean production over rented land. This puts pressure on land prices (since both kinds of producers compete for the same land) and forces traditional producers to either intensify their production or to give up for rent their plots to the investment fund managers.

The questions that are to be answered by the simulation include the following: What are the effects in the long-term over land use (system-level) of the strategies of both kinds of producers (agent-level)? How does land property evolves over time? Under what conditions do small traditional producers (those that start the simulation with fewer plots) exit the system? Which is the effect of international soybean prices in land use, either with or without the presence of investment fund managers?

Step 2: Identify Roles & Agent Types

Two (very) different agent types are identified: traditional producers and investment fund managers (which are also producers). Under the first category, three subtypes are further identified: small, medium and big traditional producers which are differentiated only by the initial amount of plots assigned (at time zero of the simulation), but all of them sharing the same traditional producer's

behaviour (rationale). Both agent types are represented by a hierarchy of concepts in the UML Class Diagram of Figure 4.

Regarding roles, each of the two agent types have distinct behaviours, which are represented by UML Activity Diagrams. For space reasons only the behavior of traditional producers (one year step) is shown in Figure 5. There is no role exchange (agent instances do not change their behavior over time) neither role multiplicity (each agent instance only plays one role at a time), so there is a simple one-to-one relationship between agent types and roles.

Step 3: Model Domain Conceptualization

The domain conceptualization is shown in the UML Class Diagram of Figure 4. It also includes the structural aspects of the environment and resources, which will be completed by adding the dynamics on step 6. This figure shows at the left-hand side the two agent types (**InvestmentFundManager** and **Traditional**) which inherit from the (abstract) concept **Producer**.

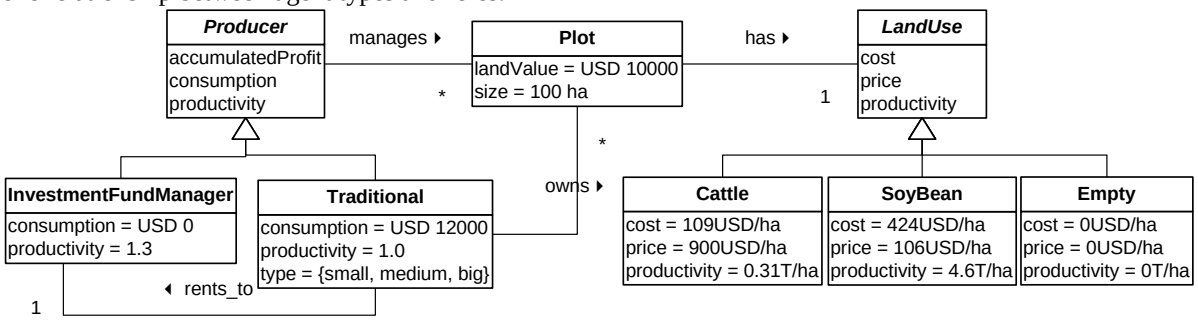


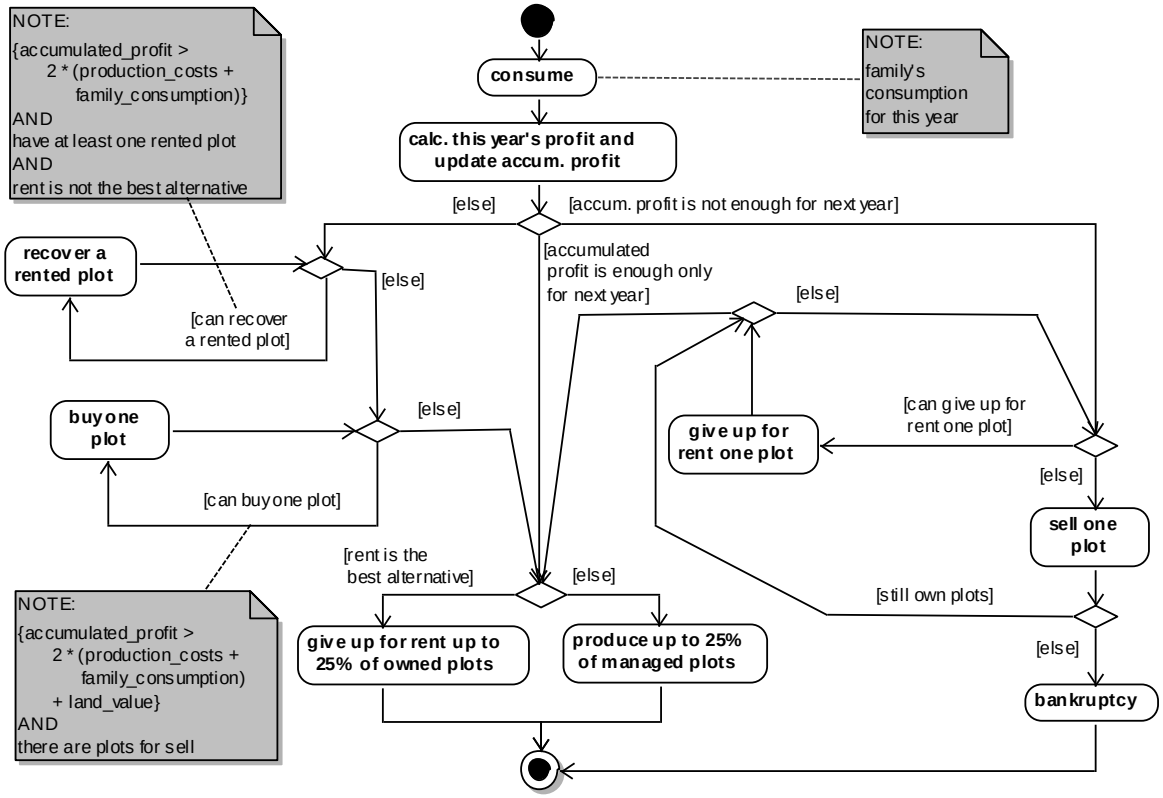
Figure 4: Domain Model Conceptualization for the Case Study.

At the middle, the concept **Plot** represents the environment which is always associated with one **LandUse**, which in turn can be **Cattle**, **SoyBean** or (temporarily) **Empty**. These three specialized concepts where identified as resources. Finally, the relation of land ownership (**owns ►**) as well as a relation for knowing who rents to who (**◄ rents_to**) and which plots are managed by which producers (**manages ►**) are represented in the figure. The figure also includes some exemplifying attributes with initial values for each concept.

Step 4: Define Agent Interaction

Figure 6 shows the agent-agent interaction that occurs when an instance of **InvestmentFundManager** is willing to rent plots to an instance of **Traditional** producer.

Figure 5: Traditional producer's behavior (at each time step).



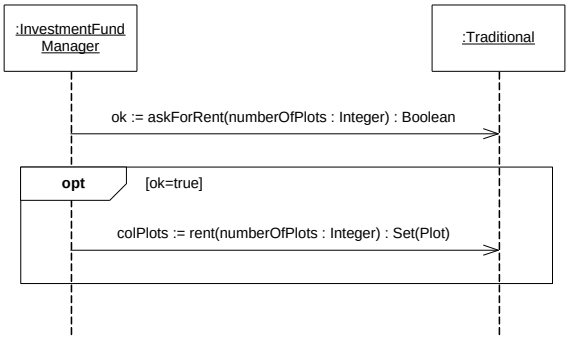


Figure 6: Example of agent-agent interaction.

Step 5: Agent Architecture & Design

This step deals with specifying the internal agent design that enables agents to fulfill their design goals. In this case, a simple reactive architecture was followed, and its design exactly corresponds to those concepts shown in the Domain Model Conceptualization of Figure 4. Therefore any other diagram showing the agent's design is omitted.

Step 6: Model Environment & Resources

Resources are not necessarily static objects that do not suffer any change or evolution over time. For example, considering cattle as a resource, and assuming reproduction as an important issue of the model, then a UML State-Transition Diagram as the one in Figure 7 could be used to represent the vital cycle of each animal. The diagram shows that cows are assumed to be born as a calf, that after one year and a half are considered as cows, that one year later are pregnant for 9 months and finally after 8 more years they exit the system (e.g. because they are sold). On the other hand, the plots (which are the environment) need no extra artifacts in this case.

Step 7: Simulation Configuration

Since this step involves the specification of several elements, all of them are briefly explained:

1) Initial Configuration: based on on-field information in the basin, 40 **Traditional** agent instances of type 'small', 20 of type 'medium' and 10 of type 'big', all of them geographically randomly assigned and with equal chance of producing cattle or soybean; and 5 instances of **InvestmentFundManager**.

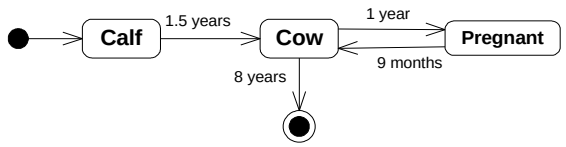


Figure 7: Cattle life cycle.

2) Time Definition: it will be a time-step simulation developed in CORMAS. Each time step represents one year, so every action taken in the simulation must be done on an early basis.

3) Task Scheduling: Figure 8 shows a possible task scheduling, where at each time step the scheduler (a component of the simulation framework software package) first lets resources evolve (e.g. calling the cattle life cycle) and then calls the agent's behaviour.

4) Input Parameters: international soybean prices are a good example of input parameters, since they are not determined nor affected by the system.

5) Output: producer's income, total number of traditional producers, total number of plots used for cattle and for soybean (land use).

6) Visualization: view land use with different colours over each cell that represents a plot (e.g. red for a cattle production and green for a soybean production).

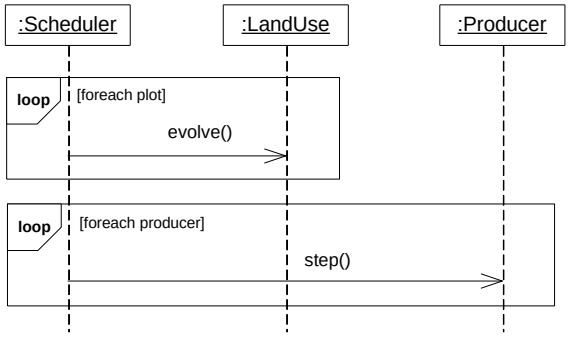


Figure 8: Task scheduling.

7 CONCLUSIONS

This article introduced a new methodological framework for developing agro-ecosystem simulations, based on the ABM approach and already existing general purpose AOSE methodologies as well as a widely used graphical modelling language as UML 2.0. Even though it is not (yet) a complete and fully comprehensive software development methodology, it has already been successfully used by our team in some research projects (not given yet for anonymity).

Although the steps were presented in a sequential order, an iterative & incremental approach may well be adopted, benefiting from user's feedback, in the same way current mainstream software development methodologies do. This would also imply that a single step would be visited more than once during the development process. Also some of the steps may change if other type of simulation is used, like event-based simulation.

Further work is ongoing in different topics with the overall aim of achieving a methodology that is of easy uptake for programmers. From our experience we believe this is a very important issue since human resources for the programming tasks are usually (very) scarce, and the more specific knowledge we require the programmers to have, the more scarce the resources becomes. This is why it is important to keep the methodology simple, using tools and techniques already familiar to programmers (like object-oriented programming, UML and iterative and incremental development).

This further work includes: a) The completeness of the methodology, including aspects such as which roles (in the development process) do what, when and how; b) Evaluate the use of specific graphical modelling languages such as AML and particularly analyze the benefits of including it against the requirement for the programmers to know it; c) Develop the necessary software tools to assist in the development process (like plug-ins to certain integrated development environment); d) Propose a semi-automatic construction process from models to simulation code, supported by the model-driven approach of the methodology; e) Continue to use and refine the methodology and get programmer's feedback about its chances to get real uptake.

REFERENCES

Cervenka. R., Trencansky, I. (2007). *The Agent Modeling Language - AML: A Comprehensive Approach to Modeling Multi-Agent Systems* (1st ed).

CORMAS (n.d.). WebSite: http://cormas.cirad.fr

CSIRO (2008). *Complex or just complicated: what is a complex system?* CSIRO fact sheet. URL: www.csiro.au/resources/AboutComplexSystems.html

Dawn, P, Hessl, A, Davis, S (2008). *Complexity, land-use modeling, and the human dimension: Fundamental challenges for mapping unknown outcome spaces*. Geoforum, Volume 39 Issue 2: 789-804.

Ellen, R. (1982). *Environment, Subsistence and Systems*. Cambridge University Press, New York.

Ferber, J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, Reading, MA.
FIPA (n.d.) Website: www.fipa.org
Gliessman, S. R. (1997). *Agroecology: ecological processes in sustainable agriculture*. Ann Arbor Press
Hecht, S. (1987). *The Evolution of Agricultural Thought*. In Altieri, M. (1987). *Agroecology: The Science of Sustainable Agriculture*. Westview Press.
Henderson-Sellers, B., Giorgini, P. (eds) (2005). *Agent-oriented Methodologies*. Hershey, PA: Idea Group.
Le Page, C., Bommel, P., (2005). A methodology for building agent-based simulations of common-pool resources management: from a conceptual model designed with UML to its implementation in CORMAS. In: Bousquet François, Trébuil Guy, Hardy Bill (eds). *Companion modeling and multi-agent systems for integrated natural resource management in Asia*. Metro Manila: IRRI, p. 327-349.
Lind, J. (2008). Website: www.agentlab.de/
Miller, J.H., Page, S.E. (2007). *Complex adaptive systems: an introduction to computational models of social life*. Princeton University Press.
Norman M. (1979): *Annual Cropping Systems in the Tropics*. Gainnesville: University Press of Florida
OMG (2005). *The Unified Modeling Language Specification v2.0*. Website: www.uml.org
Perez, P., Batten, D. (eds) (2006). *Complex Science for a Complex World, Exploring Human Ecosystems with Agents.* The Australian National University Press, Australia.
Rammel, C., Stagl, S., Wilfing, H., (2007). Managing complex adaptive systems — A co-evolutionary perspective on natural resource management. *Ecol. Econ.* 63, 9–21.
Tran, Q.N., Low, G. (2008). MOBMAS: A methodology for ontology-based multi-agent systems development. *Information and Software Technology*, 50: 697-722.
Wooldridge, M. Website (2008). *Lecture Notes on Introduction to Multiagent Systems Course*: www.csc.liv.ac.uk/~mjw/pubs/imas/teaching.html
Wooldridge, M. (2002). *An Introduction to Multiagent Systems*. Wiley & Son