UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Departamento de Ingeniería de Software e Inteligencia Artificial

METODOLOGÍA PARA EL ESTUDIO DE SOCIEDADES ARTIFICALES

MEMORIA PARA OPTAR AL GRADO DE DOCTOR
PRESENTADA POR

Candelaria Elizabeth Sansores Pérez

Bajo la dirección del Doctor:

Juan Pavón Mestras

Madrid 2007

METODOLOGÍA PARA EL ESTUDIO DE SOCIEDADES ARTIFICALES

Memoria que presenta para optar al grado de

Doctora en Informática

Candelaria Elizabeth Sansores Pérez

Dirigida por el profesor

Juan Pavón Mestras

Departamento de Ingeniería de Software e Inteligencia Artificial

Facultad de Informática

Universidad Complutense de Madrid

Junio 2007

Agradecimientos

Este trabajo nunca hubiera sido posible sin la colaboración, el apoyo y estímulo constantes de muchas personas. Estoy especialmente agradecida a mi director de tesis, Dr. Juan Pavón Mestras por su acertada orientación durante todo el proceso de los estudios de doctorado y el desarrollo de la tesis. Le agradezco la confianza que depositó en mí al acogerme en el grupo de investigación GRASIA (Grupo de Agentes Software: Ingeniería y Aplicaciones) que dirige en el departamento de Ingeniería de Software e Inteligencia Artificial de la Facultad de Informática. Agradezco su inestimable apoyo para que pudiera participar en congresos y eventos científicos, y su amistad. Su capacidad de trabajo, a pesar de otras muchas ocupaciones y dificultades, ha sido un referente para mí en el transcurso de estos años.

Beaucoup de gens ont contribué à que mon stage et expérience de recherche à Toulouse étaient inoubliables, intellectuellement et personnellement. Je remercie à toute l'équipe de recherche SMAC (Systèmes Multi-Agents Coopératifs) de l'IRIT (Institut de Recherche en Informatique de Toulouse) à l'Université Paul Sabatier à Toulouse, pour son bon accueil. En particulier, à Marie-Pierre Gleizes, responsable de l'équipe, pour m'avoir accueilli dans son équipe. Je remercie aussi à Pierre Glize pour son enthousiasme et sa présence attentive. C'était aussi inestimable l'aide de Carole Bernon qui a consacré une bonne partie de son temps à m'aider à résoudre des problèmes de langage.

También deseo expresar mi agradecimiento a las muchas personas de la Facultad de Informática de la Universidad Complutense de Madrid y, en particular, del Departamento de Sistemas Informáticos, que han colaborado en mi formación durante el doctorado, especialmente a mis profesores del curso de doctorado. A mis amigos por la alegría que me brindaron y que hicieron aún más agradable esta experiencia en España.

Sin ánimo de olvidar a nadie en particular, mi más sincero agradecimiento a todas aquellas personas que de una u otra manera han compartido mi vida durante el transcurso de estos últimos años, ya que todos forman parte de mis vivencias y mi desarrollo.

A mi madre, a la memoria de mi padre y a mis hermanos, a todos ellos gracias por su apoyo y ánimo. Gracias Mirbella por todos estos años. Nunca podré agradecerte suficientemente tu apoyo incondicional, tus consejos y amor. Aunque mucha gente me ha acompañado en este camino, sin ti nunca hubiera llegado al final.

Este trabajo de investigación ha sido financiado por el Consejo Nacional de Ciencia y Tecnología (CONACYT) de México a través del programa de becas para estudios de postgrado en el extranjero que otorga.

También ha sido desarrollado con el apoyo de la Dirección General de Universidades e Investigación de la Consejería de Educación de la Comunidad de Madrid (España) y la Universidad Complutense de Madrid (Grupo de investigación consolidado 910494) y el proyecto Métodos y Herramientas para Modelado de Sistemas Multi-agente, financiado por el Ministerio Español de Ciencia (referencia TIN2005-08501-C03-01).

Resumen

Las sociedades artificiales permiten el análisis de propiedades de complejidad que los sistemas sociales presentan, es decir, los efectos imprevisibles y agregados de una población de individuos en un mundo común. Entre estos efectos se incluyen las configuraciones y regularidades no intencionales que se encuentran en la sociedad, es decir, la emergencia de propiedades auto-organizativas que surgen de redes de individuos autónomos conectados entre si, la adaptación que los individuos pueden exhibir en su afán de auto-organizarse, y la dinámica que deriva del comportamiento de los agentes y su interacción.

Para realizar este análisis, las sociedades artificiales proponen modelar a los individuos de la sociedad como agentes autónomos y realizar el análisis del modelo mediante la simulación.

En esta tesis exploramos las diferentes metodologías y herramientas para el estudio de sociedades artificiales en el dominio de aplicación de los sistemas sociales y sugerimos que las prácticas actuales para el estudio de sociedades artificiales podrían mejorarse en tres aspectos: la metodología de desarrollo de modelos basados en agentes, el nivel de desarrollo del concepto de agente con el que se pueden modelar e implementar sistemas sociales y el alto grado de conocimiento de programación que requiere el experto del dominio social para el desarrollo de estas sociedades.

Así, el objetivo general de esta tesis es proporcionar un método de estudio y desarrollo de sociedades artificiales que facilite su implementación mediante mecanismos flexibles, que permita adoptar representaciones de agentes más complejos con propiedades intencionales, y que facilite la especificación abstracta de los modelos.

En consecuencia, la metodología plantea una ingeniería de desarrollo basada en modelos que permite trabajar en un nivel de abstracción elevado independiente de las tecnologías de implementación subyacentes. La infraestructura necesaria para la aplicación de la metodología se propone como un marco. Éste incluye un lenguaje de modelado de sociedades artificiales, que considera aspectos de intencionalidad y racionalidad de los agentes y su modelo computacional correspondiente, un mecanismo para aspectos de adaptabilidad de los agentes, y la transformación automática de los modelos a código de simulación.

En esta memoria se presenta el desarrollo de esta propuesta, las conclusiones fundamentales de la investigación y las líneas de investigación abiertas.

Extended Abstract and Conclusions

Background

A social system consists of a collection of individuals that interact among them directly or through their social environment. One of the most difficult challenges for understanding social phenomena is their intractably complex nature. The emergence of societies and the establishment of cooperative relationships among their individual components have facilitated the generation of an extremely rich repertoire of behavioural possibilities not present in solitary organisms. The collective action of social individuals produces hierarchical phenomena that extend in time and space well beyond the local environments of the participants. Large-scale processes generated by local-scale interactions without central control, so-called self-organization, may in turn affect the specification of how individuals behave, interact and respond in their most immediate spatial domains. This relationship between individual behaviour and macroscopic properties is hard to understand and experiment with.

Therefore, it is logical to consider a society as a complex system (Goldspink 2000, 2002) whose individuals can be themselves a complex system. From their interactions, there is an emergence of social phenomena that can not be identified or deduced directly from the individuals' behaviour. Consequently, the concepts of complex systems are useful and can be applied to the study of natural or artificial societies.

Social scientists have attempted to understand this complexity with different methodologies based on mathematical equations. Although these methodologies have improved social science research, they have failed in capturing emergent behaviour and self-organization, and had little success in expressing laws that regulate these phenomena (Moretti 2002). They have faced difficulties in the development of new methodologies appropriate for better exploring the complexity of social dynamics.

However, theories of self-organizing complex systems offer to the researchers of the field of social science an important theoretical and methodological framework for the study of non-linear dynamic systems. Artificial Societies (AS) are one of the perspectives adopted for the study of social dynamics.

An AS is a synthetic representation of a society (Steels 1995) modelled as a Multi-Agent System (MAS) where autonomous agents imitate society's real *actors* as well as their interactions.

A MAS (Newell 1982; Shoham 1993; Genesereth y Ketchpel 1994; Sykara 1998) consists of a set of autonomous software entities (the agents) that interact among them and with their environment, to solve problems that can not be resolved individually with the capabilities or knowledge of individual agents, meaning that they relate to achieve a goal for self-benefit or for the benefit of the community. In this same context, an agent is a software program that has *autonomy* to act without direct human or other agents' intervention, thus it can take its own decisions based on its goals and beliefs of the state of the world in which it is.

In this sense, it is understandable that the paradigm of agents has been proposed to build possible social worlds because it assimilates quite well the individual in a social system. The proposal is based in creating models usually simplified and sufficiently similar to the social phenomena and social processes that we intend to understand and that are not easily studied in a direct manner. These models as well as real phenomena, for example, the societies, are dynamic because they change in time; therefore, a model will consist not only of structure but also of behaviour. To observe a models' behaviour it is necessary the passage of time on it and it is here where *computer simulation* functionality is required.

Computer simulation consists of making computational models that are executed in a computer to simulate dynamic phenomena by means of a technique that manages their dynamic, for example, events or discrete time (Fishwick 1995b). In social sciences, computational models represent social processes, and if these models are built with agents then we talk about Agent-Based Social Simulation (ABSS). These models represent an AS that is studied by the ABSS. The ABSS allows the execution of a set of agents that are intended to describe the behaviour of observed heterogeneous social entities (e.g. individuals and organization), in an observable environment (laboratory like) where agents' behaviour can be monitored. This is one of the main advantages of AS because it allows the observation of the collective behaviour and trends of system evolution, studies that would not be feasible otherwise. We can say that the goal of the ABSS models is to study and understand the dynamics of social phenomena just like they emerge in a society of agents.

The field of research of AS, and in consequence, the one of the present investigation is the interaction of the three scientific fields mentioned previously, namely, the theory of agents, social sciences and computer simulation. It is important to notice the subtle difference between MAS and AS underlined by (Conte et al. 1998): "*If the MAS field can be characterized as the study of societies of artificial autonomous agents, agent-based social simulation can be defined as the study of artificial societies of autonomous agents*". The former treats social aspects of agents' systems and includes the study of norms, institutions, organizations, cooperation, competition, etc. In the later

the main goal is the research of the use of agents' technology to simulate social phenomena in a computer.

For the later objective many tools have been developed that facilitate the simulation of complex systems, among them social systems, using as foundation the agent paradigm.

However, its use within a research framework is not trivial if we consider that the development process involves knowledge of different experts in their own representation language in different levels: 1) the knowledge of social scientists for the abstract representation of social phenomena that involves the modelling of heterogeneous individuals as a complex structure, 2) the expert knowledge of software engineers for the design and computer representation of the latter ones.

To go from the highest level of abstraction to the implementation level, the representation has to be transformed, because different concepts are used at each level. Social scientists use information of real agents; software engineers require concepts of agents' theory for modelling and implementation. Therefore, if we do not have an adequate methodology that treats knowledge at both levels, that is, that considers modelling and implementation of AS, the simulation tools by themselves might not be sufficient to achieve this task.

Motivation

To express a theory we need a language. (Ostrom 1988) argues that there are three types of languages: natural language, mathematical language and computer language. The first one can deal with uncertainty and lack of clarity, but is ambiguous, like discursive social theories are. The mathematical language offers analytical techniques to produce derivations, but it is unlikely that it can be use to explain non-linear phenomena. Computer models provide a type of intermediate language between natural language and mathematical language; they allow formalizing methodically and unmistakably specific phenomena and are useful in situations where the processes are impossible or too difficult to formulate using mathematical functions only.

In AS the description of the actors of a real society and their behaviour are formulated with a programming language. That allows the formalization of social theories with specifications that can be transfer into computer programs. In general, the existing proposals to realize this formalization, rather than defining a specification language, are more oriented to provide a controlled simulation environment. There are different toolkits that fall into the previous classification. Java libraries that facilitate computer programmers the development of simulation environments (e.g. Repast (Repast 2004), Swarm (Swarm 2004), Ascape (Ascape 2000), Mason (Mason 2004)), rapid development environments that allow building simple models using visual programming (e.g. Starlogo (StarLogo 2004), Netlogo (NetLogo 2004), Cormas (Cormas 2004), Agentsheets (AgentSheets 2004)) and high-level modelling languages (e.g. SDML (SDML 1997), Sesam (SeSAm 2004)).

In addition to the development tools it would be also necessary to use methodologies that orient social scientists to develop simulation models as part of their theoretical or experimental research. The methodologies found in social simulation vary depending on their orientation and reach with regards to the development stages that they cover. Methodologies like those of (Fishwick 1995a; Gilbert y Troitzsch 1996; Drogoul et al. 2002; Axelrod 1997a; Goldspink 2002; Gilbert y Troitzsch 1999; Moss 1998) are a first proposal to guide the development of AS, although most of these methodological proposals are more a research methodology than a development methodology.

Despite the diversity of tools and methodologies the development of AS currently faces several challenges. These essentially refer to two main aspects, one regarding the development process and the other one the direction of the implementation tools:

- Lack of systematization in the development of AS. Although the development does not starts from scratch but rather uses the existing tools, it is remarkable the lack of research work to systematize or define a methodology for the development of useful agent-based simulations. It is not sufficient to have agent oriented development platforms to build AS.
- Lack of an agent-based high-level specification language for the design of more abstract AS models. Existing toolkits require modellers to have a good working knowledge of the programming language that they are aimed at, for example, Java or Logo. Others demand to learn a complex interface that can be as difficult to master as a full programming language, which finally limits its usability.
- Lack of a MAS model in the implementation level. Existing tools are in the middle of a dilemma. On one hand, they provide complete freedom for the development of MAS because they do not have a predefined agents' model. The programmer is the one in charge of defining and implementing the MAS as best needed, thus making the use of the tool more complex. On the other hand, some tools provide agents' models relatively easy to use. The disadvantage is that agents in this kind of tools are quite simple, usually with a poor or nonexistent model of cognitive agents, limited to interactions through the environment, like cellular automata.
- Lack of replication support of simulation models. Replication is one of the steps of the investigation process that (Axelrod 1997a) names "the distinctive mark of accumulative science". It is required to confirm that published results of a simulation are reliable in the sense that they can be reproduced by someone else.

These issues lead to first consider the need to provide a methodology for the study of AS that structure the development of agent-based social simulations in accordance to software engineering practices. That methodology would be used to guide the development of ABSS models and their

execution, starting with the representation of social phenomena in a model design, supported by a specification language; and ending with the implementation and simulation of that model like a MAS.

In addition to the methodology, it will be needed to provide a solution for the limitations that existing tools have for the development of adequate social simulations. Social scientists that want to use a methodology that structures the development of AS must confront a difficulty of practical order that should not be minimized. The use of existing simulation tools is not simple as models have to be specified as programs, where the representation of concepts that define social phenomena is not evident. This makes the definition of models a complex task for sociologists, as they usually do not have the skills for computer programming.

Therefore, a specification language at a more abstract level would be desirable. This would allow the definition of abstract models independent of programming languages or platforms. Some efforts have been made to assist the high-level modelling of this systems, like SeSAm (Shell for Simulated Agent Systems) (Klügl et al. 2004), which allows the graphical modelling of behaviour as finite state machines using UML-like activity diagrams (Unified Modeling Language) (OMG 2003b), or Repast Py, which provides assistants to generate Repast models (Lutz y Ascher 1999). However, these solutions are limited to simple applications. In order to model systems with certain complexity it is necessary to extend the provided basic behaviour libraries, which requires advance programming knowledge.

This leads us to an open debate on the complexity of simulation models. Most of the ABSS tools lack of an underlying sophisticated model of agents and MAS. Mainly because their goal is to assist social scientists in the development of simple agent models that react through changes made in their environment by other agents, interacting indirectly through the same environment.

Although the MAS paradigm seems to adapt quite well to the nature and peculiarities of social phenomena, allowing to overcome the limitations of analytical methods, and even if they are used as a modelling paradigm in the AS research method, a doubt exists on its appropriate use and if MAS potential is not being leveraged. Due to this simplicity in agents' implementation, it is natural to question whether these are really computational agents as those found in agents' theory or mere cellular automata (in many cases).

Although modelling requires simplification, the plausibility issue with regard to social simulation also arises, which, in principle, involves modelling human cognition a little more realistically. This plausibility issue demands the need of agents with reasoning capability and intentionality (goal-oriented). Social systems are not only dynamic systems composed of a high number of actors and interactions among them, but also actors are themselves complex structures whose behaviour is unpredictable (although it is possible to find AS of some natural systems that have simple

actors, like in the case of some insects, which, despite of being simple actors, are considered as societies because of their mutual interactions (Castelfranchi 1998a)).

The complexity or simplicity of agents is a critical matter in the domain of AS. Concerning simplicity, we can conclude that agent-based modelling is not about reproducing complexity but to understand it. (Axelrod 1997a) points out that complexity must be on the simulated results and not on the models' assumption. The most interesting results are obtained when complexity emerges at the macro level through simple dynamics at micro level. Concerning complexity, the idea that agents must be as realistic as possible is emphasized, that is, in the case of social simulation this likeliness refers, in principle, to modelling the mental processes of an individual in a more realistic manner (Conte 1999).

Therefore, we can realize that the level of simplification in which agents would be modelled is going to depend on the nature of the problem under study. For example, the racial segregation models of (Schelling 1978) or the cultural dissemination models of (Axelrod 1997b) use quite simple agents' models whose main goal is the local interaction among individuals and the observation of emerging behaviours at a macro level. On the other hand, in Agent-based Computational Economics, as in the example of (Pascual 2006), agents with a higher level of rationality, although not completely rational to prevent the observation of certain behaviours that characterize human beings like emotions, motivation, etc., are suggested (Lopez Paredes et al. 2002). This type of bounded rationality (Simon 1982) is associated with more realistic models of individuals with cognitive capabilities, information, perception, etc. In both cases interesting results have been obtained.

Taking into account these precedents, we must guarantee that the decision to include simple or complex assumptions in the model is a decision of the social scientist based on the expected explanation that he wishes to obtain and not due to limitations of the tools they use. Consequently, it would be ideal that the tools include a MAS model that could be adapted to the needs of the social scientists according to the level of complexity of the models.

In summary, we have identified several open issues in existing practices for the study of AS. From a practical point of view, there is a lack of a construction logic or development methodology that involves activities that go from the models' abstract representation to the correct transformation into computational agent models. Furthermore, the existing tools adopt solutions or only address a minimal part of ABSS development process. Mainly, there is a lack of tools oriented to the development (especially for analysis and design) of agent-based models.

Due to the nature of AS, the framework of study can be inspired by Agent Oriented Software Engineering (AOSE) (Jennings y Wooldridge 2000). Existing AOSE methodologies and graphic modelling languages provide a higher level of abstraction that those found in ABSS. Concepts used in AOSE can be closer to those that a social scientist would use; hence, this makes them more

appropriate to solve the issues under consideration. However, it is certain that there are still some issues to solve, for example, how to provide certain facilities that are not yet considered in AOSE because they are not usually required for the development of MAS (such as time and space issues in simulation).

Starting from this hypothesis, this investigation proposes the creation of a methodological framework for the AS study and a set of tools to aid the development. The methodology will be adapted from the practices of an agent oriented methodology. Among the existing methodologies in AOSE, this thesis specifically proposes the use of INGENIAS (Pavón and Gómez-Sanz 2003), given its characteristics for MAS modelling and the associated tools.

Objectives

The general objective of this thesis is the development of a well defined process for ABSS, which should facilitate implementation through flexible mechanisms that allow adopting more complex agent representations than those currently supported by agent based simulation toolkits. To achieve this, we propose a methodological framework for the study of AS. This framework considers a model driven engineering approach (MDE) (Schmidt 2006) and a guide that facilitates the construction and experimentation of AS. This general objective can be specified in the following ones:

1. Review of complex social systems and emergence. This revision sets the foundation for the theoretical and methodological research of complex social systems and emergence through AS.
2. Definition of a methodological process for the development and study of AS. This process will define the activities that are to be carried out to construct an AS starting from the knowledge of the domain expert. How to transform this knowledge from a more abstract level to a specific computational model will be defined by means of MDE practices. The central idea of MDE is to use models at different abstraction levels for the development of systems. In this study we will use agent-based simulation models. The main activity of the domain experts in this process would be the design of models (instead of developing code), with the guide of this methodology, which will clearly define the activities to follow. The process will specify the sequence of models to develop and how to derive a model from another that resides in a level immediately superior of abstraction. In this form, we will assure that the end user follows a development life cycle based on AOSE and that he knows what to do and how to do it at any moment. We also consider the activities that are a part of the study of AS, for example, experimentation activities.

3. Definition of an Artificial Societies Modelling Language (ASML). This language will be based on the visual modelling language of MAS of INGENIAS. It will be described using INGENIAS meta-models that will be extended to include certain concepts of social domain and simulation not considered previously in MAS. This will allow describing AS graphically in an abstraction level closer to the social domain concepts. This declarative language will facilitate to the domain experts to visually describe the simulation models, which, in principle, will be easier than with imperative programming languages. Furthermore, one of ASML objectives will be to provide flexibility to define macro aspects, such as the organizational structure of a system, and micro aspects, like simple to complex agents with intentional characteristics. Concerning ASML, we are aware of the fact that having a universal modelling and simulation language for the social domain is not feasible (Gilbert et al. 1997; Gilbert 1996). Therefore, our aim is to provide an agent oriented language that can be customized to particular social domains, through specialization or addition of new elements, which can be defined by the experts of such domains. In this sense, the use of meta-models can facilitate these extensibility and specialization requirements for the language.
4. Definition of a models transformation mechanism and code generators for simulation. The transformation mechanism is part of the techniques of MDE that promote the refinement of abstract models into more particular or specific models of a platform. To convert one model to another model of the same system it has to be defined how to perform the refinement, the additional information and the platform specific information that is necessary to integrate at the lower abstraction level. With this automatic transformation process it is possible to synthesize multiple types of artefacts: alternative representation of the models, source code, documentation, etc. In particular, code generators will be defined that allow transforming the graphic model into a computer program for the two simulation platforms chosen in the present investigation. This last proposition is interesting in order to replicate the same model on different environments and better validate the obtained results. The choice of Repast and Mason as experimentation platforms lies on their availability, their spread acceptance, and mainly the advanced simulation infrastructure they provide.
5. Definition of a MAS Computational Model for Social Simulation. Finally, to implement the transformation mechanism it would be necessary to include an abstraction level on the simulation platform libraries. This abstraction level is a library that defines a computational agent model that allows implementing intentional agents, guided by goals with adaptation capabilities through a reinforcement learning mechanism (Di M. Serugendo et al. 2006). This library is necessary due to lack of agent concepts in existing platforms that would allow maintaining a consistency between conceptual and implementation level concepts. Thus, if we were to transform actual models into implementation models in these plat-

forms, the result would be the reduction of agent concepts into computational abstractions such as objects, messages, vectors, etc. Therefore, to make sure that the agent paradigm is applied not only at design level but also at implementation level, this library, which is added to the chosen simulation languages, will be provided.

Chapters

The details of the present investigation as well as the obtained results and experimentation that make part of this thesis are gathered in the following eight chapters summarized in what follows.

Chapter 1: In this chapter we describe the background, our motivations and objectives of the present investigation.

Chapter 2: This chapter describes the context of the application field of the present investigation, that is, social systems, and considering particularities of human societies. The characteristics of these systems are presented. To do so, we review their complexity dimensions, the micro-macro link, and the emergence. Also, we examine how these systems are studied; specifically we present the study method that arises from the complexity theories, that is, AS.

Chapter 3: In this chapter we make a revision of the state-of-the-art of agent-based simulation methodologies and tools. First, the methodological proposals for an investigation process with AS are reviewed. Next, a revision is made of the commonly used methods in AS design. Finally, we examine the tools in AS implementation. Our aim is to evidence the need of an alternative perspective for the development process in AS, and determine the aspects that can be improved.

Chapter4: The proposed AS modelling language (ASML) is described. First, we justify the choice of INGENIAS as basis of the ASML. Then, we review the INGENIAS modelling language, and finally we explain the extensions made to allow the specification of ABSS models.

Chapter 5: In this chapter, a description of the support tools of the proposed methodological framework for the AS study is made. Among these we describe the Java libraries with the computational agent model proposed to extend the agent model currently available in existing simulation platforms, including an agent adaptability mechanism, and finally, the integration of all the previous tools into the simulation code generation mechanism.

Chapter 6: Here, the proposed methodology for AS development and study is illustrated. First we present the motivation that leads to the proposed methodology, then the principles on which it is based and finally, the process, roles and artefacts that have to be produced (together with a SPEM specification).

Chapter 7: This chapter illustrates the use of ASML and the process, including replication, with a case study that applies the methodology and the code generating tools proposed in this thesis. The developed system is based on the investigated conditions that lead to altruism during so-

cial interaction and how influences the modelling of individuals as rational entities. The system was inspired in a famous ethological study of altruism among bats (Wilkinson 1984).

Chapter 8: In the last chapter we discuss conclusions, limitations, suggestions for future research and a list of publications related to this research work.

Conclusions

This chapter summarizes the main contributions of this thesis: original proposals of the present work, limitations and future lines of investigation.

Introduction

The main contribution of this thesis has been a methodology for the study of AS. The methodology proposes a guide for AS development and analysis as societies of intelligent agents. This methodology is based on a model driven engineering approach, which allows working in an abstraction level that is independent of the target simulation toolkit. The needed infrastructure in order to apply this methodology is proposed as a framework. It includes an Artificial Society Modelling Language (ASML) that considers aspects of intentionality and rationality of agents and its corresponding computational model, a mechanism for the agent adaptability aspects, and a module for the automatic transformation of models into simulation code.

Our initial hypothesis in the present work was that existing practices for the study of AS are insufficient in three aspects: the methodology of ABSS development, the level of development of the agent concept in ABSS, specially when considering human societies, and the high degree of programming knowledge that the social domain expert requires for the development of this societies with current tools.

Some existing methodologies focus in the experimentation stages and neglect MAS development, specially issues concerning AS modelling. Others that consider this activity are based on rather simple agent models; therefore, the elaboration process that they propose is incomplete when used to model intelligent agent societies with mental patterns that have consequences in the development of the society. Other more advanced consider the need of a more developed MAS model; however, they do not define how to develop it.

As to the implementation tools, most of them are oriented to programming languages that provide the freedom of defining agent models adapted to each problem. Others consider a higher specification level but are rather constrained to play with simple agent models.

This makes AS modelling a complex task and, in great measure, dependable on the programming knowledge of the social domain expert. Furthermore, the used level of abstraction makes dif-

ficult the analysis of emergent social patterns that lacks a counterpart at modelling level for their interpretation. Consequently, there is a need for techniques that allow to address these issues, reinforcing the existing practices from an Agent Oriented Software Engineering (AOSE) approach.

This reasoning leaded to the necessity to extend the concepts of purely reactive agents on which existing methodologies and tools are sustained, to concepts of intelligent agents, which would facilitate the specification of human society models. As societies are formed by individuals that take decisions and act, it was necessary to use a concept of more advanced agents, with social and intentional characteristics, in line with the proposals of AOSE. Moreover, following this reasoning, it was necessary to consider a language for the abstract modelling of these intelligent agents and mechanism that facilitated its computational representation and its later simulation.

Therefore, for the accomplishment of the techniques that would address the previous aspects, the use of an existing methodology in the AOSE was considered, specifically INGENIAS. The justification for its adaptation to achieve these techniques is based on the characteristics of the MAS model that it adopts and its philosophy of model-based development. Consequently, it was possible to consider micro aspects of a social system such as agent intentional behaviour and macro aspects, such as the specification of organizational structure and its dynamics.

The adaptation of INGENIAS methodology to the study of AS, covering the above issues, has lead to the following contributions:

1. ASML language for AS visual modelling. This has been defined with a set of abstractions of the social simulation domain that were added to INGENIAS MAS modelling language. The ASML language allows model specification on a less ambiguous form than the natural language used in social disciplines and it is naturally integrated to the proposed development process. The agents' cognitive abstractions of the language, such as mental states, intentionality, autonomy, etc. allow a more realistic representation of social system models, such as human societies. In addition, the language can be extended to include new social domain concepts. Finally, the language facilitates the identification and analysis of emergent processes and social patterns in terms of the specification elements.
2. MAS computational model. The elements of this model correspond and implement the elements of the MAS conceptual model of the ASML language, with the purpose of preserving the semantics and that the resulting application is functionally equivalent to this model. Therefore, the computational model considers a MAS model of intelligent agents. This is placed over the Repast and Mason simulation platforms to endow them with a more developed agent model than the one they provide. The correspondence between the elements of both abstraction levels allows the interpretation of results in terms of the elements of the conceptual model.

3. Architecture of adaptability by reinforcement. It was designed to provide more realism to the modelling of individuals in a complex social system. The architecture provides the agents with aptitudes to dynamically modify their behaviour in accordance with some reinforcement. This behaviour is achieved by the agents' motivation to pursue certain goals. The consequence of this reinforcement on motivation is that an individual agent can adapt his aptitudes and exhibit a specialization of roles.
4. Automatic code generation and analysis module. This module deals with the automatic transformation of the model into source code. It is built with the interface that INGENIAS provides for traversing the specifications, select the information of the model that we want to implement and place the selected information in the prototypes designed for the Repast and Mason simulation platforms in which it would be later executed. This strategy of model-based development allows replication to contrast results and study the effects that the facilities of different simulation environments can have over the simulation results. For the analysis, the results are presented directly in the simulation environments; in this way, the diagrams of results provided by the simulator are easily interpreted by the user to identify social patterns in terms of the model elements. Therefore, the two first contributions are considered essential, that is, having an abstract specification and that their elements have corresponded adequately with the elements of the model code in the simulator. Additionally, these three contributions altogether allow isolating the domain expert of implementation aspects and encourage focusing in the analysis of results.
5. Methodology for the study of AS. The methodology provides a guide for the development of AS like MAS and its integration to a wider experimentation process. The model-driven engineering philosophy is the way to facilitate this task. The methodology provides a well defined process to develop MAS of intentional and rational agents, including their analysis and design, and the transformation of the AS specification into an executable simulation model. The methodology defines an iterative and incremental process for the study of AS with three phases. The phase of identification of the subject under study, the phase of AS engineering, and the experimentation phase. For each phase workflows are defined respectively identifying the following: system definition, design and platform-specific model generation, and study of emergent properties. For each workflow the methodology defines the activities and roles that participate in each activity, artefacts that produces, and tools to be used. The definition of roles in this cycle allows that each one focuses in the activities they do best. Moreover, it delimits responsibilities and improves communication between roles. With this, the domain experts can focus their effort in modelling and evaluation of results.

With these contributions the AS study is reinforced in two forms, first in the modelling and development of agents that exhibit rational, intentional and adaptability characteristics, and second, in the analysis of emergence of unpredictable and aggregated effects in these societies.

Open issues and future research

The main difficulty of this proposal is related to the specification and automatic code generation. MAS modelling involves many aspects that must be considered and its specification can be quite complex. With a visual language and a development guide the difficulty is lessen, but still prevails the level of excessive details that are required for full automatic code generation and certain aspects that the guide does not cover due to its generalist nature. To overcome this problem, our aim is to define modelling languages for specific application domains, for instance, a specific social theory that considers MAS specific aspects relevant for its application. This reduces not only the complexity of the model specification, but also the level of details that must be specified for its implementation.

Another aspect that can be improved is to extend the library of self-organization mechanisms. Different mechanisms exist that adapt better for diverse situations. The adaptability mechanism by reinforcement was well fitted for modelling the self-organization that individuals exhibit when they learn from past experiences. Nevertheless, it would be interesting to include other mechanisms to model self-organization, for instance, through the individuals' cooperative behaviour when they interact locally with others. A method that we have considered to be included is based on the AMAS theory (Capera et al. 2003; Georgé et al. 2003). This method would allow modelling individuals with self-organization capabilities, reorganizing their local interactions with other agents and the environment when they recognize a non-cooperative situation. To achieve this, they react based on their knowledge, the representations that they have of other agents and on the individual task that they have to carry out.

Regarding the AS analysis module, currently it is not provided another method for the interpretation of results than the one provided by the simulation platforms. Until now, under the experimentation frame of this study it has been sufficient. Nevertheless, in a larger framework other tools to aid the analysis can be required, for example, for the study of the models' sensitivity to the initial parameters.

Concerning the automatic code generation, we have experimented with two simulation platforms widely used in the social simulation domain. The proposal regarding the aid for the development made in this methodological framework is not limited to these two environments; therefore, it is feasible and desirable to include new environments to broaden our experimentation.

Finally, this proposal has been validated with experiments in our own research group. A wider experimentation in the context of other projects and groups remains as an interesting avenue for future research, and in a longer term, the possible exportation of the proposed techniques to other application domains.

Índice

Capítulo 1.

Introducción

El primer capítulo de esta tesis está dedicado a describir la motivación que se encuentra detrás de este trabajo de investigación así como el contexto en el que fue desarrollado y los objetivos perseguidos durante su realización. Concluye con la delineación del contenido y estructura del documento.

1.1. Contexto

Un sistema social está constituido por un colectivo de individuos que interactúan mutuamente entre sí o a través de artefactos de su entorno social. Uno de los desafíos más importantes para poder entender los fenómenos sociales reside en el estudio de la emergencia de comportamientos en las sociedades resultado de la colaboración entre sus componentes individuales. La acción colectiva de individuos sociales produce fenómenos que se extienden en el tiempo y el espacio más allá del entorno local de los participantes. Los procesos de gran escala generados por interacciones de escala local sin un control central, es decir, la denominada "auto-organización" puede en su momento afectar la especificación de cómo se comportan los individuos, cómo interactúan y cómo responden en sus dominios espaciales más inmediatos. Esta relación entre el comportamiento individual y las propiedades macroscópicas es difícil de comprender y sobre todo de experimentar con ella.

Es lógico, por lo tanto, considerar una sociedad como un sistema complejo (Goldspink 2000, 2002) cuyos individuos interactúan entre si. A su vez, ellos mismos pueden ser sistemas comple-

jos. De sus interacciones emergen fenómenos sociales que no pueden ser identificados o deducidos directamente del comportamiento de los individuos. Consecuentemente, los conceptos de sistemas complejos son útiles y pueden ser aplicados al estudio de sociedades naturales o artificiales.

Los científicos sociales han intentado comprender esta complejidad desde varias perspectivas. Una de ellas ha sido la aplicación de diferentes metodologías basadas en ecuaciones matemáticas. Aunque estas metodologías han mejorado la investigación en el campo de las ciencias sociales, también han fallado en capturar los comportamientos emergentes y la auto-organización, y han tenido muy poco éxito en expresar leyes que regulen estos fenómenos (Moretti 2002). Es decir, se han enfrentado a dificultades en el desarrollo de nuevas metodologías apropiadas para la mejor exploración de la complejidad de la dinámica social.

Sin embargo, las teorías de sistemas complejos auto-organizados ofrecen a los investigadores del campo de las ciencias sociales un importante marco teórico y metodológico para el estudio de sistemas dinámicos no-lineales. Uno de estos enfoques que se ha adoptado para el estudio de la dinámica social son las *Sociedades Artificiales* (SA).

Una SA es una representación sintética de una sociedad (Steels 1995) modelada como un *Sistema Multi-Agente* (SMA), donde agentes autónomos imitan *actores* reales de la sociedad así como sus interacciones.

Un SMA (Newell 1982; Shoham 1993; Genesereth y Ketchpel 1994; Sykara 1998) puede ser brevemente definido como una red de agentes que interactúan entre ellos y con su entorno para resolver problemas que no pueden ser resueltos de forma individual con las capacidades o conocimientos de cada agente, es decir, que se relacionan para alcanzar un objetivo en beneficio propio o de la comunidad. En este mismo contexto, un agente es un programa software que cuenta con a*utonomía* para actuar sin la intervención directa de humanos u otros agentes, esto quiere decir que puede tomar sus propias decisiones en base a sus objetivos y creencias del estado del mundo en el cual se encuentra.

Es comprensible que el paradigma de agentes sea propuesto para construir posibles mundos sociales pues se asemeja bastante bien al individuo en un sistema social. La propuesta se basa en crear modelos generalmente simplificados y suficientemente similares a los fenómenos y procesos sociales que se quieren comprender y que no son fáciles de estudiar directamente. Estos modelos al igual que los fenómenos reales, como pueden ser las sociedades, son dinámicos pues cambian con el tiempo, así un modelo consta no solo de estructura sino también de comportamiento. Para observar el comportamiento del modelo es necesario el paso del tiempo en el mismo, y es aquí donde entra en función la *simulación por ordenador*.

La simulación por ordenador consiste en la elaboración de modelos computacionales que son ejecutados en un ordenador para simular fenómenos dinámicos por medio de alguna técnica que gestiona su dinámica, por ejemplo eventos o tiempo discreto (Fishwick 1995b). En el caso de las

ciencias sociales, los modelos computacionales representan procesos sociales, y si estos modelos son elaborados por medio de agentes entonces se habla de *Simulación Social Basada en Agentes* (SSBA). Así, estos modelos representan una SA que es estudiada por medio de la SSBA. La SSBA permite ejecutar un conjunto de agentes, que describen el comportamiento de entidades sociales heterogéneas (p.ej. individuos, organizaciones), en un entorno en el cual se pueden realizar observaciones de su comportamiento, como si fuera un experimento controlado en un laboratorio. Esta última es una de las principales ventajas de las SA pues permiten realizar estudios que de otra forma serían inviables, permitiendo además la observación del comportamiento emergente y la evolución del sistema. Finalmente, se puede perfilar que el objetivo de los modelos de SSBA es estudiar y comprender la dinámica de fenómenos sociales tal y como surgen en una sociedad de agentes.

El área de investigación de las SA, y en consecuencia de este trabajo de investigación, es la intersección de los tres campos científicos mencionados anteriormente, esto es, la teoría de agentes, las ciencias sociales y la simulación por ordenador. Por supuesto, es importante notar la sutil diferencia entre SMA y SA remarcado por (Conte et al. 1998): "*Si el campo de los SMA puede caracterizarse como el estudio de (o implementación de) sociedades de agentes autónomos artificiales, la simulación social basada en agentes puede definirse como el estudio de sociedades artificiales de agentes autónomos*". En el primero se tratan aspectos sociales de sistemas de agentes e incluye el estudio de normas, instituciones, organizaciones, cooperación, competición, etc. En el segundo, el objetivo principal es la investigación del uso de la tecnología de agentes para la simulación de fenómenos sociales en un ordenador.

Para este último propósito se han desarrollado varias herramientas que permiten la simulación de sistemas complejos, entre ellos los sistemas sociales, utilizando como base el paradigma de agentes software.

Sin embargo, su utilización dentro de un marco de investigación no es trivial si consideramos que el proceso de desarrollo involucra el conocimiento de diferentes expertos con su propio lenguaje de representación en diferentes niveles: 1) el conocimiento de los científicos sociales para la representación abstracta de fenómenos sociales, que involucra el modelado de individuos heterogéneos como una estructura compleja, 2) el conocimiento experto de informáticos para el diseño y representación computacional de los mismos.

Para ir del nivel más abstracto al nivel de implementación la representación debe transformase, pues en cada nivel se usan conceptos diferentes. Los científicos sociales utilizan información de agentes reales, los informáticos requieren conceptos de la teoría de agentes para su modelado e implementación. Por lo tanto, si no se cuenta con una metodología adecuada que integre el conocimiento de ambos niveles, es decir, que considere tanto el modelado como la implementación de SA, las herramientas de simulación por sí mismas son insuficientes para llevar a cabo esta tarea.

1.2. Motivación

Para poder expresar una teoría, es necesario un lenguaje. (Ostrom 1988) argumenta que hay tres tipos de lenguajes: lenguaje natural, lenguaje matemático, y lenguaje computacional. El primero puede tratar con incertidumbre y falta de claridad, pero es ambiguo, como suelen ser las teorías sociales discursivas. El lenguaje matemático ofrece técnicas analíticas para producir derivaciones, pero difícilmente puede usarse para explicar fenómenos no-lineales. Los modelos computacionales proporcionan un tipo de lenguaje que está a la mitad del camino entre el lenguaje natural y el matemático, permiten una formalización cuidadosa e inequívoca de un fenómeno específico y son útiles en situaciones donde los procesos no pueden representarse usando únicamente modelos matemáticos.

En el caso de las SA, actualmente la descripción de los actores de la sociedad real y su comportamiento se realiza con un lenguaje de programación. Ello permite formalizar las teorías sociales con especificaciones que pueden ser trasladadas a programas de ordenador. En general, las propuestas existentes para realizar esta formalización (véase la sección 3.3), más que definir un lenguaje de especificación, están más orientadas a proporcionar un entorno de simulación controlado. Existen varios conjuntos de herramientas que caen en la clasificación anterior. Librerías Java que permiten a los programadores construir entornos de simulación (p.ej. Repast (Repast 2004), Swarm (Swarm 2004), Ascape (Ascape 2000), Mason (Mason 2004)), entornos de prototipado rápido que permiten la construcción de modelos simples utilizando programación visual (p.ej. Starlogo (StarLogo 2004), Netlogo (NetLogo 2004), Cormas (Cormas 2004), Agentsheets (AgentSheets 2004)) y lenguajes de modelado de alto nivel (p.ej. SDML (SDML 1997), Sesam (SeSAm 2004)).

Además de las herramientas de desarrollo, ha sido necesario también valerse de metodologías que orienten a los científicos sociales a desarrollar modelos de simulación como parte de sus investigaciones teóricas o experimentales. Las metodologías encontradas en la simulación social varían según su orientación y alcance en cuanto a las etapas del desarrollo que cubren (véase sección 3.2). Las metodologías como las de (Fishwick 1995a; Gilbert y Troitzsch 1996; Drogoul et al. 2002; Axelrod 1997a; Goldspink 2002; Gilbert y Troitzsch 1999; Moss 1998) son una primera propuesta para guiar el desarrollo de SA, aunque en su mayoría estas propuestas metodológicas se acercan más a lo que es una metodología de investigación que una metodología de desarrollo.

Pese a la diversidad de herramientas y metodologías propuestas, el desarrollo de SA adolece hoy día de ciertos inconvenientes. Éstos se refieren esencialmente a dos aspectos principales, uno en cuanto al proceso de desarrollo y otro a la propia orientación de las herramientas de implementación:

- *Falta de sistematización en el desarrollo de SA*. Aunque el desarrollo no parte de cero sino que se utilizan las herramientas existentes, es notable la falta de trabajo para sistematizar o definir una metodología para el desarrollo de simulaciones útiles basadas en agentes. El disponer de plataformas de desarrollo orientadas a agentes no es suficiente para generar SA.
- *Falta de un lenguaje de especificación de alto nivel orientado a agentes para el diseño de modelos de SA de forma más abstracta*. Las herramientas actuales requieren que los modeladores tengan un buen conocimiento del lenguaje de programación para el cual fueron desarrolladas, por ejemplo, Java o Logo. Otras exigen aprender a utilizar una interfaz compleja que puede ser tan difícil de dominar como un lenguaje de programación completo, lo cual finalmente limita su utilidad.
- *Falta de un modelo de SMA a nivel de implementación.* Las herramientas actuales se encuentran en medio de una disyuntiva. Por un lado proporcionan completa libertad para la creación de los SMA pues no cuentan con un modelo de agentes predefinido. El usuario es el encargado de definir e implementar el SMA como mejor le convenga, siendo más compleja la utilización de la herramienta. Por otro lado, algunas herramientas proporcionan modelos de agentes que son relativamente fáciles de usar. El inconveniente es que los agentes en este tipo de herramientas son demasiado sencillos, generalmente con modelos cognitivos pobres o inexistentes, limitados a interacciones a través del entorno, básicamente autómatas celulares.
- *Falta de soporte para la replicación de modelos de simulación*. La replicación es uno de los pasos del proceso de investigación que (Axelrod 1997a) denomina "*el sello distintivo de la ciencia acumulativa"*. Es necesaria para confirmar que los resultados publicados de una simulación sean confiables en el sentido que puedan ser reproducidos por alguien más.

Estos inconvenientes conducen primero a considerar la necesidad de proporcionar una metodología para el estudio de SA que estructure el desarrollo de simulaciones sociales basadas en agentes de acuerdo con las prácticas de ingeniería del software. Dicha metodología servirá de guía al desarrollador desde la representación del fenómeno social en un modelo de diseño, mediante algún lenguaje de especificación, hasta la implementación y simulación de dicho modelo como un SMA.

Además de una metodología, para el desarrollo de simulaciones sociales adecuadas habrá que procurar una solución a los inconvenientes que presentan las herramientas en la actualidad. Los científicos sociales potencialmente interesados en utilizar una metodología que estructure el desarrollo de SA se seguirán enfrentando a una dificultad de orden práctico que no debe obviarse. La utilización que éstos hacen de las herramientas de simulación no es sencilla ya que los modelos de fenómenos sociales se especifican directamente como programas de ordenador, en los cuales la re-

presentación de los conceptos que definen los fenómenos sociales no es evidente. Esto hace que la especificación de modelos por parte de los expertos sociales sea una tarea complicada, ya que habitualmente no disponen de la adecuada formación informática para llevarla a cabo.

Por lo tanto, sería deseable un lenguaje de especificación de mayor nivel de abstracción. Esto permitiría definir modelos abstractos independientes de lenguajes de programación o de plataformas. Algunos esfuerzos se están realizando para tratar de facilitar el modelado a alto-nivel de estos sistemas, como en Sesam (Klügl et al. 2004), que permite modelar gráficamente el comportamiento como máquinas de estados finitos usando diagramas de actividad tipo UML (*Unified Modeling Language*) (OMG 2003c), o Repast Py, que proporciona asistentes para generar modelos de Repast (Lutz y Ascher 1999). Estas soluciones, sin embargo, están bastante limitadas a aplicaciones sencillas. Si se quieren modelar sistemas con cierta complejidad suele ser necesario extender las librerías de comportamientos básicos proporcionadas, lo cual requiere conocimientos de programación avanzados.

Esto nos lleva a un debate abierto relativo a la complejidad de los modelos de simulación. La mayoría de las herramientas de SSBA carecen de un modelo sofisticado de SMA subyacente, pues el objetivo primordial de estas herramientas es asistir a los científicos sociales en la realización de modelos de agentes sencillos que reaccionan a través de cambios realizados al entorno por otros agentes, interactuando indirectamente a través del mismo entorno.

Aunque el paradigma de SMA parece adaptarse de forma adecuada a la naturaleza y peculiaridades de los fenómenos sociales, permitiendo superar las limitaciones de los métodos analíticos, y aunque se utilizan como paradigma de modelado en las SA como método de investigación, existe una duda sobre la adecuada utilización y explotación de su potencial. Debido a esta simplicidad en la implementación de agentes, es natural cuestionarse si éstos son en realidad agentes, en el sentido de la teoría de agentes o la Inteligencia Artificial Distribuida, o meros autómatas celulares (en muchos casos).

Aunque el modelado requiere cierta simplificación, se presenta la cuestión de la plausibilidad con respecto a la simulación social. Ello involucra el modelar entidades sociales de forma suficientemente realista. Esta plausibilidad exige la necesidad de agentes con capacidad de razonamiento e intencionalidad (orientados a objetivos). Los sistemas sociales no son solamente sistemas dinámicos compuestos de un gran número de actores y de interacciones sino que los actores mismos son estructuras complejas con comportamiento imprevisible (aunque también es posible encontrar SA de algunos sistemas naturales con actores sencillos, como algunos insectos, que no obstante son consideradas sociedades pues interactúan socialmente entre ellas (Castelfranchi 1998a).

La complejidad o simplicidad de los agentes es una cuestión radical en el dominio de las SA. Con respecto a la simplicidad, se puede concluir que en el modelado basado en agentes no se trata de reproducir la complejidad sino de comprenderla. (Axelrod 1997a) señala que la complejidad

debe estar en los resultados simulados y no en las asunciones del modelo. Los resultados más interesantes se obtienen cuando la complejidad al nivel macro emerge a través de dinámicas simples al nivel micro. Con respecto a la complejidad, se enfatiza la idea de que los agentes deben de mantenerse tan realistas como sea posible, es decir, en el caso de la simulación social esta verosimilitud se refiere en principio a modelar los procesos mentales de un individuo de forma más realista (Conte 1999).

Se puede entender, por lo tanto, que dependiendo del problema de estudio será el nivel de simplificación con el que se modelen los agentes. Por ejemplo, los modelos sobre la segregación racial de (Schelling 1978) o el de diseminación cultural de (Axelrod 1997b) utilizan modelos de agentes muy simples cuyo principal objetivo es la interacción local de los individuos y la observación del macro comportamiento emergente. Por otro lado, en la Economía Computacional Basada en Agentes, como en el ejemplo de (Pascual 2006) se sugieren agentes con un nivel de racionalidad más elevado pero no completamente racionales que impidan la observación de ciertos comportamientos característicos de los seres humanos como emociones, motivaciones, etc. (López Paredes et al. 2002). Este tipo de racionalidad limitada (Simon 1982) se asocia a modelos de individuos con capacidad cognitiva, de información, percepción, etc. más reales. En ambos casos se han obtenido resultados interesantes.

Dados estos precedentes, debemos garantizar que la decisión de incluir asunciones simples o complejas en el modelo sea una decisión del científico social en base a las expectativas de explicación que desea obtener y no por limitaciones de las herramientas utilizadas. Consecuentemente, sería ideal que las herramientas contasen con un modelo de SMA que se pudiera adaptar a las necesidades de los científicos sociales según el nivel de complejidad de los modelos.

Como resultado de la problemática expuesta anteriormente, se puede asumir que las prácticas actuales para el estudio de SA son insuficientes en varios aspectos. Desde un punto de vista práctico, hay una carencia de una lógica de construcción o una metodología de desarrollo que involucre actividades que vayan desde la representación abstracta de los modelos hasta su correcta transformación a modelos computacionales de agentes. Además las herramientas existentes adoptan soluciones o facilitan solamente una mínima parte de este proceso de desarrollo: hacen falta herramientas más orientadas al desarrollo de modelos basados en agentes.

Dada la naturaleza de las SA, el marco para su estudio puede estar inspirado en la *ingeniería del software orientada a agentes* (Jennings y Wooldridge 2000). En esta disciplina de la Inteligencia Artificial existen metodologías de desarrollo de SMA y lenguajes gráficos de modelado de mayor nivel de abstracción que los encontrados en la SSBA. Los conceptos utilizados en esta disciplina pueden ser más cercanos a los que utilizaría un científico social y por ello parece ser más apropiada para resolver el problema que se ha planteado anteriormente. Aunque es cierto que existen al-

gunas cuestiones a resolver, como por ejemplo proporcionar algunas facilidades que no se habían tenido en cuenta, por no ser inicialmente necesarias para el desarrollo de SMA.

Partiendo de esta hipótesis, esta investigación propone la creación de un marco metodológico para el estudio de SA y un conjunto de herramientas de ayuda al desarrollo. La metodología será adaptada de las prácticas de una metodología orientada a agentes. Entre las metodologías existentes en la ingeniería del software orientada a agentes, esta tesis plantea concretamente el uso de INGENIAS (Pavón y Gómez-Sanz 2003), por sus características para el modelado de SMA y las herramientas que proporciona.

1.3. Objetivos

El objetivo general de esta tesis es que el estudio y desarrollo de SA pueda seguir un proceso bien definido y facilitar su implementación mediante mecanismos flexibles que permitan adoptar representaciones de agentes más complejos con propiedades intencionales (Newell 1982). Para ello se propone un marco metodológico para el estudio de SA. Este marco plantea una ingeniería de desarrollo basada en modelos (*Model Driven Engineering*, MDE) (Schmidt 2006) y una guía que facilita la construcción y experimentación de SA. Este objetivo general puede concretarse más en los siguientes:

1. *Estudio de los sistemas sociales complejos y la emergencia*. Este estudio sienta las bases de la investigación teórica y metodológica de los sistemas sociales complejos y la emergencia a través de las SA.
2. *Definición de un proceso metodológico para el desarrollo y estudio de SA*. Este proceso definirá las actividades que han de llevarse a cabo para construir una SA partiendo del conocimiento del experto del dominio. El cómo transformar este conocimiento de un nivel más abstracto a un modelo computacional concreto será definido por medio de las prácticas de la ingeniería MDE. La idea central de la MDE es utilizar modelos en diferentes niveles de abstracción para el desarrollo de sistemas. En este caso se trataría de modelos de simulación con agentes. La actividad de los expertos del dominio en este proceso sería diseñar modelos (en lugar de desarrollar código) guiados por esta metodología, la cual definirá claramente los pasos a seguir. El proceso especificará la secuencia de modelos que se van a desarrollar y cómo derivar un modelo de otro que se encuentra en un nivel de abstracción inmediatamente superior. De esta forma, se asegurará que el usuario final siga un ciclo de vida de desarrollo basado en la ingeniería del software orientada a agentes y sepa qué hacer y cómo hacerlo en todo momento. También se consideran las actividades que forman parte del estudio de las SA, como las actividades de experimentación.

3. *Definición de un Lenguaje de Modelado de SA (LMSA)*. Este lenguaje se basará en el lenguaje de modelado visual de SMA INGENIAS. Será descrito usando los meta-modelos de INGENIAS que serán adaptados para incluir ciertos conceptos del dominio social y de la simulación no considerados previamente en los SMA. Esto permitirá describir gráficamente las SA en un nivel de abstracción más cercano a los conceptos del dominio social. El beneficio que puede proporcionar este lenguaje declarativo a los expertos del dominio es describir los modelos de simulación visualmente, en principio más fácilmente que un lenguaje imperativo de programación. Además, uno de los objetivos de este lenguaje será el de proporcionar flexibilidad para definir aspectos macro, como la estructura organizativa de un sistema, y aspectos micro, como agentes sencillos a complejos con características intencionales. Con respecto al *LMSA*, somos consientes del hecho de que no es factible tener un lenguaje universal de modelado y de simulación para el dominio social (Gilbert et al. 1997; Gilbert 1996). Por esto, nuestro objetivo es proporcionar un lenguaje que pueda ser personalizado para dominios sociales concretos, a través de la especialización o la adición de nuevos elementos, los cuales podrían ser definidos por los expertos de tales dominios.
4. *Definición de un mecanismo de transformación de modelos y generadores de código para simulación*. El mecanismo de transformación es parte de las técnicas de la ingeniería MDE que promueve el refinamiento de modelos abstractos a modelos más concretos o específicos de una plataforma. Para este proceso de convertir un modelo en otro modelo del mismo sistema se ha de definir cómo se lleva a cabo el refinamiento, qué información tanto adicional como de la plataforma es necesario integrar en el nivel de abstracción de más abajo. Con este proceso de transformación automatizado es posible sintetizar varios tipos de artefactos: representaciones alternativas de modelos, código fuente, documentación, etc. En concreto, para la síntesis de código fuente se definirán generadores de código que permitan transformar el modelo gráfico en un programa ejecutable para las dos plataformas de simulación elegidas para este trabajo de investigación. Esto último es interesante para replicar el mismo modelo sobre distintos entornos y validar mejor los resultados obtenidos. La decisión de utilizar las plataformas existentes se debe a que consideramos más interesante reutilizar la infraestructura de simulación que éstas proporcionan, la cual es muy avanzada.
5. *Definición de un modelo computacional de SMA para Simulación Social*. Finalmente, para poder implementar el mecanismo de transformación será necesario incluir un nivel de abstracción sobre las librerías de las plataformas de simulación. Este nivel de abstracción es una librería que define un *modelo computacional de agentes* que permite implementar agentes intencionales, guiados por objetivos con capacidad de adaptación por medio de mecanismos de aprendizaje por refuerzo (Di M. Serugendo et al. 2006). Esta librería es necesaria, ya que si lleváramos a cabo la transformación de modelos abstractos a modelos de

implementación en el estado actual de estas plataformas, el resultado sería la reducción de conceptos de agentes a abstracciones computacionales tales como objetos, mensajes, vectores, etc., pues estas plataformas carecen de conceptos de agentes que permitan mantener una consistencia entre los conceptos del nivel conceptual y el de implementación. Por lo cual, para asegurarnos de que el paradigma de agentes se aplique no solo a nivel de diseño sino también a nivel de implementación, se proporcionará esta librería que se añade a los lenguajes de simulación elegidos.

1.4. Estructura del documento

Los detalles de esta investigación, así como los resultados obtenidos y la experimentación que conforman esta tesis se recogen en los siguientes capítulos que se resumen a continuación.

El segundo capítulo se enfoca a describir el contexto del campo de aplicación de este trabajo de investigación, esto es, los sistemas sociales. Se presenta cómo se caracterizan estos sistemas. Para ello se hace una revisión de sus dimensiones de complejidad, el vínculo micro-macro, y la emergencia. También se revisa cómo se estudian estos sistemas, específicamente se presenta el método de estudio que surge de las teorías de la complejidad, esto es las SA.

El tercer capítulo hace una revisión del estado del arte de metodologías y herramientas para la simulación basada en agentes. Primero se repasan las propuestas metodológicas para un proceso de investigación con SA. Después se hace una revisión de los métodos generalmente utilizados en el diseño de SA. Finalmente, se revisan las herramientas de implementación de SA. La revisión de estos enfoques tiene como fin evidenciar la necesidad de un enfoque alternativo para el proceso de desarrollo de SA, y determinar qué aspectos pueden mejorarse.

El cuarto capítulo describe el lenguaje propuesto para el modelado de SA. Primero se justifica la utilización de INGENIAS como base del lenguaje *LMSA*. Posteriormente se presenta el lenguaje de modelado INGENIAS para luego describir las extensiones que se le han realizado para permitir la especificación de modelos de SSBA.

El quinto capítulo esta dedicado a describir las herramientas de apoyo del marco metodológico para el estudio de SA propuesto. Entre éstas se describen las librerías Java con el modelo computacional de agentes propuesto para las plataformas de simulación, el mecanismo de adaptabilidad para los agentes, y finalmente la integración de todas las herramientas anteriores en el mecanismo para la generación de código de simulación.

El sexto capítulo presenta la metodología para el desarrollo y estudio de SA. Primero se presenta la motivación que da lugar a esta propuesta metodológica, después se presentan los principios

sobre los que se basa, y finalmente se describe el proceso, roles, y los artefactos que se han de producir.

El séptimo capítulo presenta la replicación de un caso de estudio que aplica la metodología propuesta en esta tesis y las herramientas para la generación de código. El sistema desarrollado se basa en la investigación de las condiciones que dan lugar al altruismo durante la interacción social y cómo influye el modelado de los individuos como entidades racionales. El sistema está inspirado en un famoso estudio etológico sobre el altruismo entre murciélagos (Wilkinson 1984).

Finalmente, el octavo capítulo presenta las conclusiones obtenidas en este trabajo de investigación y las líneas de investigación futuras que da lugar

Capítulo 2.

Sistemas Sociales y Sociedades Artificiales

Los sistemas sociales son sistemas inherentemente complejos que se han estudiado durante largo tiempo por medio de diferentes métodos analíticos. Sí bien estos métodos han permitido el avance de esta ciencia, por otro lado han demostrado ciertas limitaciones a la hora de representar las diferentes dimensiones de la complejidad de los fenómenos sociales, tal como la complejidad de los individuos, su heterogeneidad, la complejidad de sus interacciones, y la dinámica social observable en diferentes escalas de tiempo. Por otro lado, la estructura de un sistema y las leyes que rigen su comportamiento pueden representarse por medio de símbolos de algún programa de ordenador el cual permite emular el sistema real durante su ejecución. Esta técnica permite analizar sistemas complejos que de otra forma son imposibles o muy difíciles de estudiar con métodos matemáticos solamente. En este capítulo se hace una revisión de la emergencia en sistemas sociales y su relación con la complejidad. La comprensión de este espacio de investigación es primordial para estudiar el método de análisis que surge de estas dos disciplinas: la simulación de sociedades basada en modelos de agentes.

2.1. Introducción

El campo de aplicación de la propuesta que hace esta tesis está dedicado a los sistemas sociales, teniendo en cuenta aspectos cognitivos de los individuos, como en el caso de las sociedades humanas.

Un sistema social está constituido por un colectivo de individuos que interactúan mutuamente entre sí o a través de su entorno social de forma dinámica. Los fenómenos sociales que surgen de esta interacción son absolutamente contingentes, y por tanto impredecibles y cambiantes lo cual contribuye a la complejidad de estos sistemas.

La intención de explicar la naturaleza de un sistema social como un sistema complejo siempre se ha considerado en las ciencias sociales. Por ejemplo, en el siglo dieciocho los científicos de la época comparaban a las sociedades con mecanismos artificiales complejos como los relojes, y en los años 60s y 70s la *teoría general de sistemas* (Bertalanffy 1968) se basaba en la premisa de que los sistemas complejos en todos los niveles de análisis, incluyendo a las sociedades, podrían comprenderse usando el mismo conjunto de teorías y metodologías (Sawyer 2005).

Actualmente, la ciencia de la complejidad resulta de gran utilidad como metáfora de estudio de los sistemas sociales. En esta ciencia, en términos muy generales, los fenómenos complejos son aquellos cuyo comportamiento es difícilmente predecible para los efectos de las interacciones entre sus componentes.

Por su parte, los sistemas sociales se ajustan muy bien a la definición anterior. Por un lado las leyes que rigen su comportamiento pueden ser muy simples, pero los resultados de la interacción social pueden ser completamente impredecibles. Es decir, aún si se pudiera tener una completa comprensión de los factores que afectan la acción individual, esto no sería suficiente para predecir el comportamiento del grupo que forman los individuos.

Extendiendo la teoría de los sistemas complejos a las ciencias sociales se puede decir entonces que los sistemas sociales, como por ejemplo las sociedades humanas, son configuraciones complejas de muchos individuos comprometidos unos con otros en patrones traslapados y entrelazados de relaciones que son capaces de evolucionar y de adaptarse. Cada individuo por si mismo es un sistema complejo que además es consciente de si mismo y de su entorno, lo cual agrega un nivel más de complejidad.

Consecuentemente, en base a la afirmación anterior, las dimensiones de complejidad que se encuentran en los sistemas sociales son las relacionadas con la dinámica de estos sistemas y la contingencia e indeterminación de sus procesos como resultado de la evolución de los actores del sistema, sus relaciones y su heterogeneidad.

Para el estudio de los sistemas sociales y sus dimensiones de complejidad la formulación matemática ha tenido poco éxito. Por ejemplo, las simulaciones con *dinámica de sistemas* (Forrester

1971) capturan las propiedades macro de la sociedad y sus modelos simulan características globales del sistema como la población, el índice de pobreza, densidades urbanas, etc., pero fallan en capturar micro propiedades como la toma de decisiones del individuo, su heterogeneidad y sus interacciones, todo esto necesario para estudiar la complejidad del sistema.

Como alternativa para el modelado de estos aspectos micro se plantea la *simulación basada en sistemas multi-agentes*. Esta técnica permite simular sociedades usando distintos agentes computacionales, uno por cada individuo de la sociedad, y sus relaciones sociales pueden representarse por medio de interacciones entre agentes. De esta forma se crean sociedades artificiales (SA) para ejecutar experimentos virtuales en los cuales las propiedades de los agentes se hacen variar con el tiempo para observar los cambios subsecuentes en el macro-comportamiento del sistema observado. Así, las SA se han adoptado como una forma de modelar y comprender los procesos sociales y la relación entre el individuo y el colectivo, la cual es una de las cuestiones fundamentales en el estudio de estos sistemas que da lugar a los debates de mayor importancia como son el *vínculo micro-macro* y la *emergencia*.

Puesto que esta investigación busca el desarrollo de un marco metodológico para el estudio de SA, se considera conveniente hacer una exploración de los sistemas sociales desde un punto de vista de los debates que suscita y de su complejidad, la cual está directamente relacionada con la dificultad para modelar estos sistemas. Por lo tanto, la comprensión de esta complejidad representará un avance en el diseño de soluciones para tratar estos sistemas, específicamente el modelado con sistemas multi-agente.

El capítulo se estructura de la siguiente manera. La sección 2.2 hace una introducción de los sistemas sociales. La sección 2.3 introduce la teoría de los sistemas complejos, específicamente los sistemas complejos adaptativos cuyo método de estudio se adopta para el análisis y comprensión de los sistemas sociales. La sección 2.4 presenta las SA y sus aspectos metodológicos, de modelado, y simulación. Finalmente, la sección 2.5 plantea una serie de conclusiones respecto al potencial y posibles limitaciones de las SA como herramienta de estudio de los sistemas sociales complejos.

2.2. Sistemas Sociales

En general, las definiciones de sistemas sociales se encuentran entre dos extremos, las que definen los sistemas sociales como resultado de la interactividad de los individuos participantes (Watkins 1955), y las que los definen como unidades abstractas a priori, es decir, con carácter ontológico (Durkheim 1964). La primera, considera a los individuos y al colectivo participante como objetos fundamentales de interés. En la segunda, los individuos son simplemente miembros del sistema social. Recientemente, se ha considerado la idea de la *emergencia* para resolver este deba-

te. En estas ideas como las de (Kontopoulos 1993; Archer 1995) se acepta que lo único que existe son los individuos y sus interacciones pero que los sistemas sociales a los que dan lugar no pueden estudiarse en base a sus componentes, es decir que los fenómenos sociales una vez que han emergido son autónomos y son una realidad social que necesita analizarse desde una perspectiva global.

Por lo tanto, la definición de lo que es un sistema social depende de la perspectiva bajo la que se observe. Lo que es evidente es que tanto el colectivo de individuos como sus interacciones mutuas o a través de artefactos de su entorno social son fundamentales para la existencia de estos sistemas.

Con respecto a los sistemas sociales, lo que es de interés para este trabajo de investigación es cómo interacciones simbólicas sucesivas entre individuos autónomos resultan en la emergencia de fenómenos colectivos, es decir, comprender las micro-interacciones sociales y cómo contribuyen a la emergencia social por medio de modelos de simulación. Para llevar a cabo el estudio de estos fenómenos, con modelos que imiten su comportamiento, es necesario comprender cómo se estudian y cuál es la complejidad que se habrá de enfrentar para su modelado.

Una sociedad es un conjunto de individuos autónomos que comparten fines, conductas, cultura y representaciones simbólicas y que se relacionan interactuando entre sí, cooperativamente, para constituir un grupo o una comunidad situados en un entorno. Las sociedades de humanos se estudian por las llamadas disciplinas sociales, principalmente la Sociología y otras como la Antropología y la Economía. En un sentido aún más amplio, se habla de sociedad virtual cuando se habla de fenómenos que se generan y observan en grupos bajo interacción en el ciberespacio, SA como las de agentes software interactivos o sociedades de robots, de autómatas, de criaturas digitales, etc.

El estudio sistemático de las sociedades, la acción social y los grupos que la conforman, lo lleva a cabo la *Sociología*. Esta ciencia social estudia cómo son creadas, mantenidas o cambiadas las organizaciones y las instituciones que conforman la estructura social, el efecto que tienen en el comportamiento individual y social, y los cambios en éstas, producto de la interacción social. La Sociología aplica métodos de investigación empíricos, análisis de datos, elaboración de teorías y valoración lógica de los argumentos. Es la rama del conocimiento que hace de las relaciones humanas su objeto de estudio, aplicando de modo sistemático la razón y la observación e integrando explicación teórica y verificación empírica.

La Sociología trata de explicar cómo funcionan las sociedades a través de diferentes teorías o perspectivas. Las perspectivas teóricas son diferentes formas de percibir y comprender el mundo social. Por consiguiente, la característica distintiva de la Sociología no es tanto lo que es estudiado sino cómo es estudiado y explicado. Los hechos sólo se presentan a sí mismos como tales en respuesta a un intento de los individuos de comprender y explicar el mundo. Por ejemplo, el hecho de que un pedazo de madera es de un metro de largo y no solamente un pedazo de madera deriva no de la madera en si, sino de un ‘modelo’ o ‘teoría’ de medida, inventado por seres humanos. Así, al final, un ‘hecho’ es esencialmente un producto de la interpretación teórica elegida.

2.2.1. El Vínculo Micro-Macro

Las diferentes perspectivas sociológicas generalmente tienden a caer en dos categorías diferentes: Sistemas Sociales (Macro Sociología) y Acción Social (Micro Sociología), la relación entre éstas es conocida como el *vínculo micro-macro* (Alexander y Giesen 1987; Huber 1991; Knorr-Cetina y Cicourel 1981; Ritzer 2000).

En la perspectiva de los sistemas sociales, o estructuralismo (Caws 2000), se considera que el individuo nace dentro de un sistema social en curso que es independiente (del individuo mismo) y determina su comportamiento. El individuo actúa de acuerdo al "guión" establecido por la sociedad. Los valores, instituciones y cultura de la sociedad determinan las acciones y los roles, y éstos son adquiridos en el proceso de socialización. Esta es una propuesta donde el punto de partida son las sociedades como un todo y la forma en que éstas determinan el comportamiento humano.

Por su parte, en la perspectiva de acción social, un ser humano tiene pensamiento consciente. La acción humana no es simplemente una reacción a los estímulos externos sino resultado de significados, teorías, motivos, e interpretaciones traídos a una situación social por el individuo. La realidad social es una propiedad "constantemente emergente", no es algo fijo e inevitable. Esta es una propuesta voluntarista que enfatiza las acciones voluntarias de los individuos, haciendo énfasis, por ejemplo, en el 'libre albedrío'.

Ahora bien, existe una relación entre ambas perspectivas que trata de explicar cómo fenómenos macro-estructurales emergen de acciones individuales. Este aspecto *micro-macro* es conocido como *emergencia*. Por otro lado, se encuentra la *causalidad social,* que explica la influencia de los fenómenos macro-estructurales en el individuo, es decir, *macro-micro*. Y, por último, la dialéctica entre la emergencia y la causalidad social, esto es, cómo la macro-micro causalidad influye en el proceso de micro-macro emergencia. Estos tres aspectos son conocidos actualmente como el *vínculo micro-macro* de la teoría sociológica.

Un ejemplo claro del proceso emergencia y causalidad social que se puede observar en las sociedades son las naciones. Una nación es una sociedad que emerge de los individuos, sus costumbres y tradiciones. Una vez que la nación ha emergido, y se establecen leyes por medio de su constitución, ésta tiene un efecto causal en los individuos que la conforman, digamos que "rige" su comportamiento dictando leyes y normas de actuar "correctas" a las que normalmente se adhieren.

2.2.2. La Emergencia

La emergencia se refiere a la aparición de leyes, patrones u orden, a través de los efectos cooperativos de las sub-unidades de un sistema complejo. Así, el fenómeno emergente o las leyes no son una propiedad intrínseca de las sub-unidades, sino más bien algo que es una propiedad del sistema

como un todo. Un ejemplo simple es la temperatura, que a nivel microscópico no tiene ningún sentido, pero es una característica del sistema mismo.

En las ciencias sociales el concepto de emergencia permite evaluar y reconciliar las varias concepciones (descritas por el *vínculo micro-macro*) de la relación individuo-sociedad, como por ejemplo el macro orden que surge de procesos del micro nivel. No obstante, la idea de la emergencia en las ciencias sociales es tratada desde distintos puntos de vista, que guardan relación con teorías filosóficas como el *atomismo reduccionista*, el *holismo* y el *emergentismo*.

El *atomismo reduccionista* considera que la única forma científica de comprender un sistema complejo es primero analizar sus partes o sus componentes, después descubrir las reglas y leyes que describen estos componentes, y finalmente analizar las interacciones entre las partes (Nagel 1961). Sin embargo, una de las características más distintivas de la ciencia de la complejidad contemporánea es que rompe con la tradición reduccionista.

La teoría tradicionalmente opuesta al atomismo ha sido el *holismo*, que mantiene la posición de que hay fenómenos sistémicos complejos que deben ser estudiados en sus propios términos, que los métodos reduccionistas no son aplicables a tales sistemas, y que ninguna parte puede comprenderse excepto en su relación con el sistema completo (Phillips 1976). El problema con que se enfrentan los holistas es cómo basar de forma ontológica su antireduccionismo. Es decir, si el holismo acepta la posición ontológica del materialismo, esto es, que solamente la materia física existe, entonces es contradictorio su argumento antireduccionista, pues un fenómeno no es nada más que la materia de sus componentes.

Por otro lado, el *emergentismo* es una forma de no-reduccionismo que acepta la posición ontológica del materialismo. Con respecto a los fenómenos complejos bajo estudio, el emergentismo acepta que solo existen las partes componentes y sus interacciones (y no el sistema como un todo), de esta forma evade los problemas ontológicos del holismo. Rechaza el atomismo arguyendo que el reduccionismo no es una consecuencia necesaria del materialismo y que algunos fenómenos complejos naturales no pueden estudiarse con métodos reduccionistas. Estos fenómenos son sistemas complejos en los cuales estructuras de "más alto nivel", más complejas y diferenciadas, emergen de la organización e interacción de componentes más simples de "más bajo nivel".

Estas ideas filosóficas son la base de las posturas sociológicas para la comprensión de las propiedades emergentes de los sistemas sociales:

1. *Individualismo Metodológico*. El planteamiento reduccionista en Sociología considera que todos los fenómenos sociales pueden y deben ser explicados en términos de individuos y relaciones entre individuos, y que no hay propiedades sociales irreducibles. Los macro fenómenos son explicados por propiedades micro y el comportamiento de los individuos. Esta postura exige que todos los eventos sociales (p.ej. la inflación) en la teoría social sean

analizables en términos de los intereses, actividades, etc. de los individuos que los provocan y que son miembros de la sociedad (Watkins 1955).

En lo que respecta a la emergencia, se acepta la existencia de propiedades sociales emergentes aunque tales propiedades pueden reducirse a explicaciones en términos de individuos y sus relaciones. El enfoque de los individualistas metodológicos en los procesos micro-macro es considerado explícitamente como un estudio de cómo las propiedades sociales emergen de la acción individual ((Axelrod 1997c; Coleman 1987, 1990; Epstein y Axtell 1996; Homans 1964)). Así, se puede decir que los *individualistas emergentistas*, como se clasifican en (Sawyer 2001), creen que las propiedades y leyes macro-sociales emergentes pueden explicarse en términos de propiedades y leyes sobre los individuos y sus relaciones. La observación de que algunos sistemas complejos manifiestan propiedades de más-alto nivel emergentes que no son atribuibles a los componentes no es por si misma incompatible con el reduccionismo. Los individualistas metodológicos sostienen que aunque se pudieran identificar leyes acerca de grupos sociales que no puedan reducirse a leyes sobre individuos en el presente, éstas son solamente explicaciones provisorias e incompletas. Asume que con el avance del conocimiento científico se podrá, a la larga, efectuar la reducción a leyes sobre individuos.

2. *Holismo Metodológico*. La perspectiva holista, considera la sociedad como una realidad por si misma que no puede deducirse de contextos individuales (esto es, de la acción y el comportamiento). En esta postura la sociedad no obedece a la suma de sus partes. Los sujetos individuales juegan un papel secundario en la construcción de la vida social y sus condiciones actuales. El comportamiento social de los individuos debe explicarse en términos de posiciones o funciones de estos individuos dentro de un sistema social y las leyes que lo gobiernan. El principal interés del holismo metodológico es el análisis de la sociedad como un todo, por medio de sus estructuras sociales objetivadas. Una argumentación que explica esta postura la proporciona Durkheim (Durkheim 1964). Ésta enfatiza la naturaleza externa de las instituciones sociales, expresando que los fenómenos sociales son externos a los individuos y que las representaciones sociales pueden examinarse independientemente de los individuos que forman la sociedad.

   Con respecto a la emergencia, el holismo metodológico se plantea que la emergencia es incompatible con el reduccionismo individualista, y que aunque la estructura social o las propiedades emergentes (instituciones, organizaciones, sistemas sociales, cultura, etc.) son dependientes de las acciones de los individuos, esta estructura es irreducible a ellos y autónoma de forma ontológica, es decir, que es una realidad social. La autonomía aquí significa que ningún individuo tiene el poder de cambiar estas estructuras e igualmente difícil será para un grupo de individuos. También significa que la estructura no es dependiente de

la existencia de un individuo específico, la estructura sobrevive al individuo. A las teorías sobre la emergencia que se sustentan en la postura del holismo metodológico se les conoce como *colectivistas o realistas emergentistas*.

Algunos ejemplos de emergencia que pueden observarse en los sistemas sociales son:

- *Segregación social*. La emergencia de grupos con atributos comunes como: localización, cultura, estilo de vida, riqueza, clase social, etc.
- *Dinámica de opinión*. La emergencia de grupos de opinión común mediante el efecto de la influencia social en una población.
- *La evolución del lenguaje*. La emergencia de representaciones comunes. Los lenguajes cambian frecuentemente a través del tiempo, tanto el vocabulario como la gramática. Tales cambios no son concientemente seleccionados por ningún organismo oficial, no son impuestos por la fuerza a la población, sino más bien son un fenómeno emergente, que surge de las conversaciones del día a día en pequeños grupos, esparcidos a través de la sociedad.
- *Organizaciones de negocio*. Emergen de las actividades de sus empleados, además de las acciones de grupos como legisladores, abogados, anunciantes y proveedores.
- *Mercados*. Emergen de las acciones individuales de los comerciantes.
- *Instituciones religiosas*. Emergen de las acciones de sus adeptos.

Una inquietud que puede tenerse al analizar las sociedades es cómo identificar si los fenómenos que se observan son emergentes o no. En algunos casos, se dice que las propiedades de un sistema son emergentes cuando son *impredecibles*, en otros cuando son *irreducibles* y aún más, en otros casos cuando son *originales*, es decir, cuando no las posee ningún componente del sistema (Cilliers 1998). En general, un fenómeno es emergente si requiere nuevas categorías para describirlo, las cuales no son requeridas para describir el comportamiento de los componentes subyacentes (Gilbert 2002).

Finalmente, las metodologías actuales para el análisis de los sistemas sociales adoptan esencialmente alguna de estas posturas de forma poco rigurosa. Por ejemplo, las simulaciones con sistemas multi-agente modelan a los individuos y sus interacciones, y proporcionan explicaciones en base a las propiedades emergentes observadas durante la simulación. Por otro lado, las simulaciones con sistemas dinámicos modelan las características globales de la estructura social y evalúan los aspectos generales del sistema independientemente de los individuos.

2.2.3. Dimensiones de Complejidad

Los fenómenos sociales son altamente complejos, pues se hallan imbricados en complejas redes de interacción e interdependencia mutua. Un sistema social está constituido por un colectivo de in-

dividuos que interactúan mutuamente entre sí o a través de artefactos de su entorno social. Estos individuos evolucionan de forma autónoma (tienen su idiosincrasia particular), están motivados por sus propias creencias y objetivos personales, y las circunstancias del entorno social en el que se mueven moldean en gran medida dichas creencias y objetivos personales.

Esas creencias y objetivos pueden a su vez evolucionar en el tiempo para cada individuo, bien sea por su peculiar peripecia biográfica, irrepetible, o por el momento que atraviesa en su ciclo vital o por los cambios de las tendencias sociales (cambios en el entorno).

Por otro lado, es preciso tener en cuenta que todos los fenómenos sociales son absolutamente contingentes, y por tanto impredecibles y cambiantes. No están sujetos a leyes, sino a tendencias, las cuales pueden afectar a los individuos probabilísticamente. La indeterminación de los procesos y los sistemas sociales es mucho mayor que en los sistemas físicos o incluso los biológicos.

Todos estos determinantes contribuyen a que un sistema social sea altamente dinámico y complejo y por ello su reductibilidad mediante el mero recurso a la modelización matemática (modelización mediante ecuaciones estructurales, análisis estadísticos multivariantes o tratamientos estadísticos de series temporales) adolece de serias limitaciones, tanto desde el interés explicativo como predictivo. El principal problema de la modelización matemática es que sólo permite jugar con tendencias (probabilísticas) y no considera el comportamiento original del sujeto, pieza básica de cualquier sistema social. Los efectos dinámicos de procesos altamente retroalimentados no quedan bien reflejados en los tratamientos matemáticos-estadísticos, y sin embargo dichos procesos son consustanciales a los sistemas sociales reales.

Así, en los sistemas sociales se pueden identificar varias dimensiones de complejidad relacionadas con:

1. *Los individuos*. Los individuos son entidades racionales moldeados por su propia idiosincrasia que persiguen satisfacer sus propios objetivos. Su comportamiento está motivado por sus creencias y objetivos, y las circunstancias del entorno social en el que se mueven influyen en su comportamiento. Las creencias y objetivos pueden a su vez evolucionar en el tiempo para cada individuo, ya sea por cambios en el entorno social o por nuevos deseos o como resultado de generalizaciones de experiencias que ha guardado en su memoria. En el seno de las sociedades humanas, los individuos tienen capacidad de razonamiento. Las personas tienen la capacidad de reconocer, razonar y reaccionar a las propiedades emergentes, es decir son reflexivas. No sólo son capaces de distinguir patrones de acción colectiva sino que sus acciones pueden ser influenciadas por la existencia de esos patrones. La *emergencia* a la que da lugar este proceso reflexivo es la que se conoce como *emergencia de segundo orden*. Es decir, esta emergencia surge en las sociedades humanas pues los individuos son capaces de reconocer propiedades emergentes y éstas pueden influir en su comportamiento, por ejemplo una institución emergente de segundo orden es la Iglesia, la cual surge

de la interacción de los individuos, los cuales la reconocen y a su vez su comportamiento se puede ver influenciado por ésta.

2. *La heterogeneidad de los individuos*. Los individuos que forman las sociedades varían grandemente en sus capacidades, deseos, necesidades, conocimiento, cultura, estilo de vida, a diferencia de otros sistemas que están compuestos de unidades similares sino es que idénticas.
3. *Red social*. La interacción entre los individuos implica que sus preferencias, información, elecciones, objetivos, etc. son afectadas por el comportamiento de otros individuos. Los tipos de interacciones por si mismos pueden ser complejos, como la cooperación, la emoción, comunicación, etc. Esta última añade mayor complejidad cuando se lleva a cabo por medio de un lenguaje, especialmente en el caso de los sistemas humanos. Por medio de las interacciones se va formando una red de relaciones que puede llegar a ser bastante compleja en su estructura espacial y social. Los patrones que se forman son bastante intrincados, involucrando diferentes espacios como pueden ser vecindarios, naciones, etc. y diferentes relaciones sociales como las relaciones de negocio, de amistad, etc. Los patrones que surgen de estas redes sociales son tan importantes para comprender estos sistemas, que se han propuesto técnicas para analizarlas como el análisis de la centralidad, análisis de posición, difusión a través de la red, la teoría de grafos aleatorios, etc. (Carrington et al. 2005; Wasserman y Faust 1994),. El "análisis de redes sociales" por tanto es un conjunto de instrumentos para conectar el mundo de los actores (individuos, organizaciones, etc.) con las estructuras sociales emergentes que resultan de sus relaciones.
4. *Complejidad de la dinámica social*. Las sociedades son el resultado de procesos dinámicos. Sus individuos están en constante movimiento, viven, actúan llevando a cabo actividades, escuchando, hablando, reproduciéndose, etc., y reaccionan al entorno. La sociedad emerge de estos cambios constantes y solamente se mantiene mientras sus miembros conserven esta dinámica. La complejidad de la dinámica se refleja a la hora de estudiar y observar estos sistemas, pues los cambios en los individuos y el efecto de estos cambios en la sociedad son invisibles en ciertas escalas de tiempo, por lo que para considerarlas habría que llevar a cabo observaciones por largos períodos de tiempo.

2.3. Teoría de la Complejidad

El estudio de los sistemas complejos no se sustenta solamente en una teoría, por lo tanto no existe una teoría unificada de lo que es la complejidad. Más bien, la teoría de la complejidad se refiere al estudio de los sistemas complejos desde alguno de los marcos teóricos propuestos.

La Figura 1 muestra el desarrollo histórico de las propuestas que han surgido para el estudio de diferentes sistemas complejos (Goldstein 1999). Éstas se encuentran clasificadas en tres categorías, denotando su origen más general: las ciencias naturales (incluyendo biología, química, física y matemáticas), cibernética y la teoría general de sistemas, y la corriente de los sistemas complejos adaptativos.

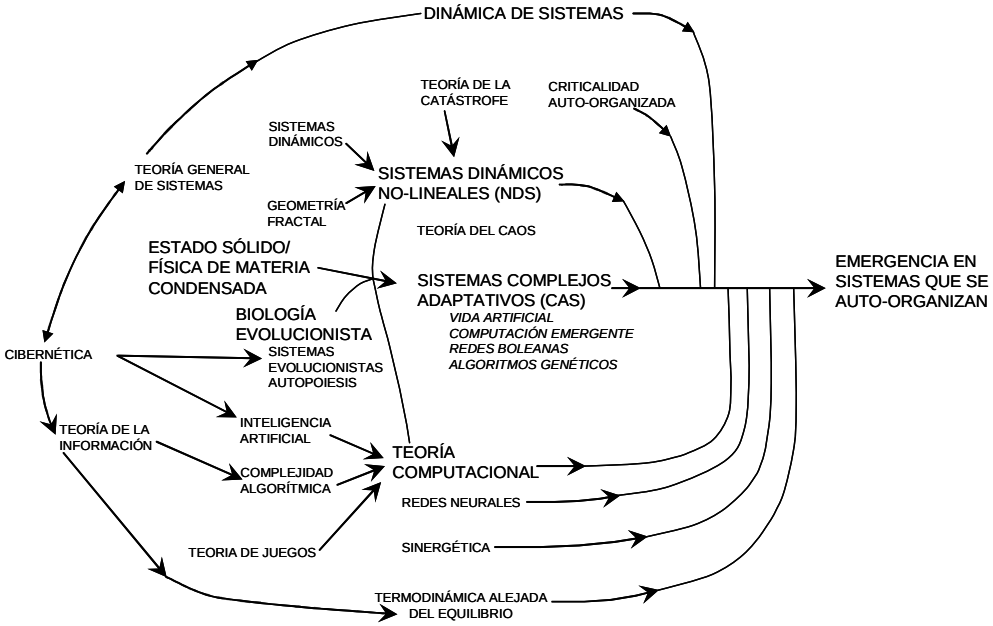

Figura 1. Desarrollo histórico de la ciencia de la complejidad y la emergencia (Goldstein 1999)

Algunas de las disciplinas y sus aportaciones más importantes al estudio de los sistemas complejos que se pueden observar en la Figura 1 son: la biología evolutiva (Darwin 1859; Fisher 1958), la termodinámica y el análisis de la evolución compleja a través de los *sistemas alejados del equilibrio que tienden* a *auto-organizarse* (Nicolis y Prigogine 1977, 1989; Prigogine 1980), la *cibernética de "segundo orden"* (Von Foerster 1981, 1996) que proporcionan los fundamentos para una teoría del observador, la *autopoiesis* (Varela 1979; Zeleny 1981) y la capacidad de los sistemas de producirse a sí mismos, la *dinámica de sistemas* (Forrester 1961) y sus modelos de simulación dinámicos, la *teoría de juegos* (Von Neumann y Morgenstern 1953) y su utilidad para comprender el efecto global de "reglas" locales, y la *Inteligencia Artificial* con sus técnicas de solución de problemas que los sistemas adpatativos (personas, organizaciones, ordenadores, etc.)

suelen sobrellevar con la complejidad (Simon 1996; Simon 1962) y su enfoque en el diseño de máquinas inteligentes.

En general, los sistemas complejos en todos estos enfoques se caracterizan por presentar las propiedades estructurales y de comportamiento siguientes:

- Constan de *muchos componentes que interactúan* entre sí, y exhiben *comportamientos* que no son una consecuencia *obvia* de la interacción entre sus componentes. Además, son *dinámicos* y *no-lineales.* Por tanto, poseen un *comportamiento* que suele tener una fuerte *dependencia de las condiciones iniciales* del sistema, lo que lo hace *difícilmente predecible*.
- Presentan *propiedades emergentes o emergencia*, el fenómeno por el cual propiedades globales de un sistema emergen de las interacciones entre sus elementos. Estas propiedades se le adjudican solamente al sistema y no son presentadas por sus elementos individualmente. Por ejemplo, una neurona no tiene inteligencia. Pero billones de neuronas interactuando entre sí pueden dar cabida a la mente, la cual es emergente. Así, la información contenida en el sistema en conjunto es superior a la suma de la información de cada parte analizada individualmente.
- Tienen tendencia hacia la *auto-organización* y a la *formación de patrones*, es decir, tienden en su desarrollo hacia mayor orden, organización y complejidad. Exhiben una remarcable *adaptación* a circunstancias cambiantes, usando mecanismos de *retroalimentación* y análisis *paralelo* de las opciones. Por tanto, se encuentran en balance entre el *orden* y *desorden*, pues aunque los sistemas complejos tienden a ordenarse, sus componentes no llegan a estar completamente en su lugar, este es el punto conocido como el *borde del caos* (Langton 1990) (un tipo de orden de movimiento impredecible).
- Los sistemas complejos con frecuencia exhiben *histéresis*. Es decir, los sistemas complejos tienen una memoria, la historia de un sistema complejo puede ser importante. Los sistemas complejos son sistemas dinámicos que cambian con el tiempo, y estados previos pueden tener una influencia en estados presentes.
- Aunque se conozca de manera precisa el comportamiento de las partes constituyentes de los sistemas complejos (determinista), no es posible anticipar con certeza su evolución futura (*no-determinista* o aleatorio). De esta forma, los sistemas complejos se encuentran entre estos dos extremos.

2.3.1. Sistemas Complejos Adaptativos

De las teorías de la complejidad mencionadas anteriormente, el enfoque conocido como *Sistemas Complejos Adaptativos* (*Complex Adaptive Systems*, CAS) exhibe propiedades que caracterizan muy bien un sistema social. Es por esto que su método de estudio se ha adoptado como méto-

do para el estudio de estos sistemas. La propuesta de los CAS (Waldrop 1992; Gell-Mann 1994; Casti 1994) se distingue por el uso extenso del modelado y simulación con SMA como herramienta de investigación.

Así, un CAS es un sistema de agentes individuales, que tienen la libertad de actuar de maneras que no son siempre totalmente predecibles, y cuyas acciones están interconectadas de tal forma que las acciones de un agente cambian el contexto para otros agentes. Estos sistemas son llamados adaptativos para enfatizar el aprendizaje y la adaptación que ocurre en ellos.

En un CAS, los agentes operan de acuerdo a sus propias reglas internas o modelos mentales. Los agentes en un CAS pueden compartir modelos mentales, o ser totalmente individualistas y heterogéneos. Más aún, pueden cambiar sus modelos mentales. Como los agentes pueden cambiar y compartir modelos mentales, un CAS puede aprender; y su comportamiento puede adaptarse con el tiempo.

El comportamiento de un CAS emerge, como en los sistemas complejos, de las interacciones entre los agentes. El comportamiento del CAS es no-lineal, aparentemente pequeños cambios pueden resultar en mayores variaciones en el comportamiento del sistema, mientras que aparentemente grandes cambios podrían no tener efecto alguno. Como no puede predecirse, de forma fiable, el comportamiento de un CAS a través del análisis, se debe dejar al sistema ejecutarse para observar su comportamiento y comprenderlo.

De las interacciones dentro de un CAS puede emerger naturalmente orden, éste no necesita imponerse desde afuera. Aún mas, el control se encuentra disperso a través de las interacciones entre los agentes, no se necesita un controlador central. Como un sistema complejo que es, un CAS es inherentemente auto-organizativo.

Generalmente, la mayor preocupación en los modelos para el estudio de los CAS es el comportamiento a nivel macro que emerge de las asunciones sobre las acciones e interacciones de los agentes individuales. Por lo tanto, las simulaciones por ordenador son construidas modelando directamente (grandemente simplificadas) las características e interacciones de los agentes del sistema que se está estudiando.

2.3.2. Sistemas Sociales como CAS

Los sistemas sociales poseen las propiedades que presentan los CAS. La caracterización de los sistemas sociales como CAS les permite adoptar el método de estudio de éstos últimos para el análisis de los fenómenos sociales. Así, el estudio de estos fenómenos, aplicando el modelado y simulación con SMA, consistirá prácticamente en construir SA como una herramienta de análisis y experimentación. La Tabla 1 muestra características de los sistemas sociales que les identifican como *sistemas complejos adaptativos*.

Área	Economía	Epidemiología	Finanzas
Agente	Consumidores	Población susceptible	Inversores
Heterogeneidad	Gustos, ingresos	Factores de riesgo	Información de preferencias de riesgo
Organización	Familias, firmas	Grupos sociales	Creadores de mercados de fondos mutuos
Adaptación	Efectos de la publicidad, educación	El evitar la infección o propagación	Aprendizaje
Retroalimentación	Comercio de compra, venta	Propagación de la enfermedad	Éxito o fracaso
Dinámica	Ajuste de precios	Propagación de la enfermedad	Movimientos del precio de las acciones
Comportamiento Emergente	Inflación, desempleo	Epidemias	Movimientos del mercado

Tabla 1. Sistemas sociales complejos y sus características como CAS (CSCS 2005).

En (Axtell 2000) se plantean varios motivos por los cuales emplear SA para modelar sistemas sociales, y se argumenta que dependiendo de la complejidad del modelo social, existen tres usos distintos de la simulación basada en agentes: modelos con agentes como simulaciones clásicas (cuando los modelos pueden formularse y resolverse completamente), agentes artificiales como complemento a la formulación de teorías matemáticas (para modelos parcialmente resolubles), y computación con agentes como herramienta de análisis (cuando los modelos son intratables o probablemente sin solución).

En este trabajo se tratan los dos últimos casos, por ser los que más interés tienen desde la perspectiva de los sistemas sociales complejos, ya que se trata de estudiar el comportamiento emergente de las sociedades de agentes, permitiendo la gestión de los modelos en varios niveles, tanto individual (los agentes) como colectivo (la sociedad u organizaciones de agentes). Así, en el segundo caso, cuando los modelos matemáticos puedan formularse pero no resolverse completamente, los modelos basados en agentes pueden esclarecer significativamente la estructura de la solución, ilustrar las propiedades dinámicas del modelo, sirven para probar la dependencia de los resultados de parámetros y suposiciones, y ser la fuente de numerosos ejemplos. El tercer caso surge porque existen clases de problemas importantes para los cuales, aunque es posible describir un modelo, hay algunos parámetros y configuraciones que no son estables o que pueden cambiar de muchas maneras, lo cual hace extremadamente difícil el análisis. En estos casos, la simulación basada en agentes puede proporcionar más información sobre el comportamiento del sistema.

2.4. Sociedades Artificiales

En un sentido general, una SA es una representación *sintética* de una sociedad (Steels 1995). Una representación sintética es un sistema artificial construido para conocer más acerca de un fe-

nómeno concreto. Este sistema ayudará a explicar el fenómeno sí es capaz de reproducir lo que se ha observado. Por ejemplo, bajo la perspectiva de este método, se puede conocer más acerca de la inteligencia si construimos un sistema artificial inteligente en vez de crear solamente teorías sobre ésta.

La lógica de este método sintético se representa en el diagrama de la Figura 2. En este método se generaliza o abstrae un fenómeno observado para producir una teoría. Esta teoría es usada para crear un sistema artificial, un substituto del sistema real. Este sistema artificial es operado, y su comportamiento es observado, el cual justifica o niega la teoría dependiendo de que tan similar sea este comportamiento observado de los hechos del fenómeno observado.

La idea de este método es construir un sistema artificial paralelo que debe comportarse de forma similar al sistema real del cual fue inspirado. Si lo hace, entonces ayudará a comprender mejor el sistema real.

En el método sintético de la Figura 2 se puede observar que el proceso de *ingeniería* juega un papel primordial en la construcción del sistema artificial. Existen diferentes y muy diversas propuestas para realizar este proceso. En el caso del estudio de los sistemas sociales complejos se ha justificado anteriormente que este proceso de ingeniería se realice con modelos computacionales basados en SMA.

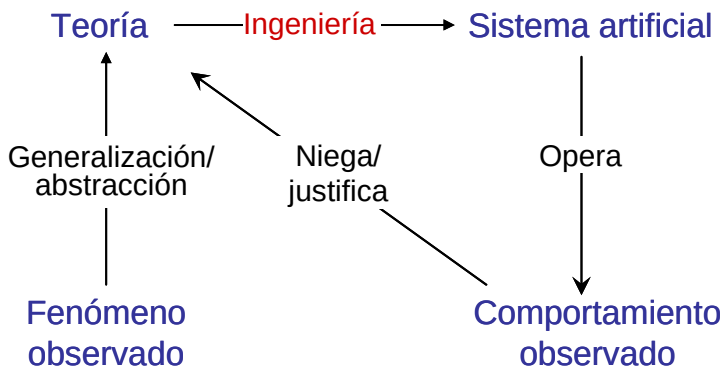

Figura 2. El Método Sintético y los Sistema Artificiales (Steels 1995)

Los aspectos fundamentales de este proceso de ingeniería involucran el modelado como parte de la construcción del sistema artificial así como la simulación para la operación del mismo, estos dos aspectos se especializan aún más cuando se involucran los SMA como paradigma de modelado.

2.4.1. Modelado

Los modelos comprenden *estructura* y *comportamiento* (p.ej. UML). Con el paso del tiempo la estructura cambia y ese cambio es el comportamiento. El modelado consiste en crear una abstracción de ambos mediante algún tipo de especificación, que puede ser textual, visual, etc. Algunas consideraciones que han de tomarse en cuenta a la hora de crear esta abstracción se revisan a continuación.

Para crear un modelo, primeramente se determina cuál es el fenómeno del mundo real que es de interés estudiar, es decir, el objeto de estudio. El siguiente paso será construir un *modelo* de este objeto de estudio, en principio más simple de estudiar que el fenómeno mismo.

La similitud del modelo con el objeto de estudio es una cuestión interesante, pues un mismo fenómeno puede modelarse de diferentes maneras, dando lugar a diferentes modelos que podrían ser suficientemente similares al objeto de estudio. Lo ideal es empezar con un modelo provisional y refinarlo poco a poco comparando los resultados obtenidos con el sistema real.

También es necesario tomar en cuenta el propósito del modelo, pues su nivel de "similitud" con respecto al sistema real dependerá de su uso potencial. Podría decirse que hay dos usos extremos. Por un lado, los *modelos de predicción*, los cuales deben producir predicciones cuantitativamente correctas dependientes de sus valores de entrada, por el otro, los *modelos para la explicación*, para los cuales los resultados cuantitativamente significativos son suficientes para comprender la reacción del sistema a valores de entrada determinados (Elzas et al. 1989). No obstante, estos dos extremos no son determinantes. En realidad existen formas intermedias de modelos con diferentes propósitos, su selección dependerá del problema de estudio.

En el caso de modelos de sistemas complejos, como los sistemas sociales, la predicción es prácticamente imposible, en gran parte debido a su imprevisibilidad. Por lo tanto, los modelos explicativos son los más viables para su comprensión. Estos modelos permiten llevar a cabo experimentos cuya motivación es la de estudiar las consecuencias de ciertas asunciones o de establecer la posibilidad de cierto resultado dado un conjunto de asunciones. Puede decirse que es una forma de teorizar sobre algún fenómeno específico.

La utilidad o la relevancia de los resultados de un modelo se evalúan dependiendo del propósito del modelo. En el caso de los modelos de predicción, su mayor preocupación es la fidelidad o proximidad de los valores previstos a los valores actuales del sistema objetivo. En el caso de modelos más teóricos o de experimentación, el requisito habitual es el de robustez. Aún así, esta evaluación no es una tarea fácil, la *validación* del modelo con frecuencia involucra comparar el comportamiento del modelo con el funcionamiento del sistema objetivo, en el caso de modelos de predicción. En el caso de modelos de experimentación, se compara con lo que es pronosticado por teorías más amplias que incluyan la que se esta probando o con otros modelos abstractos del mis-

mo sistema objetivo. Cabe mencionar que la verificación y validación de modelos de simulación es por si mismo un tema de estudio (Axtell et al. 1997; Moss et al. 1997; Schreiber 2002; Rand et al. 2003).

2.4.2. Simulación

Por su parte, la simulación es el proceso por el cual se puede observar el comportamiento del *modelo* de un sistema objetivo particular a lo largo del tiempo. La simulación permite experimentar con un modelo "ejecutándolo" bajo ciertas circunstancias de interés, observar qué es lo que pasa, con lo cual se podrá probar teorías acerca de éste. La simulación empieza en un estado inicial dado, y continúa con la repetición de la dinámica prescrita por el modelo a través de su trayectoria de tiempo.

Si la prescripción del modelo se realiza con un proceso computacional en un ordenador, entonces se habla de simulación por ordenador. La simulación por ordenador es la disciplina encargada de la construcción de modelos computacionales de un sistema existente o artificial, de su posterior ejecución en un ordenador y del análisis de los resultados de la ejecución. Una simulación cuenta con *entradas* proporcionadas por el investigador las cuales son los atributos necesarios para que el modelo coincida con el objeto de estudio, *salidas* obtenidas mientras el proceso de simulación se ejecuta y por las cuales se puede observar el comportamiento del modelo a través de una *función que avanza el tiempo*, es decir su dinámica (Fishwick 1995a). Por lo tanto, los tres pasos que generalmente constituyen la lógica de la simulación estándar son: el *diseño*, la *ejecución*, y el *análisis del modelo*. La Figura 3 muestra una vista abstracta de un modelo de simulación por ordenador.

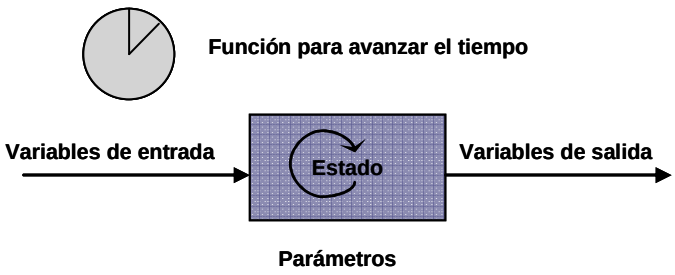

Figura 3. Principales elementos de un modelo de simulación por ordenador

La lógica de la simulación empieza con la construcción del modelo o su diseño. Para esto se cuenta con los datos recopilados mediante la observación del fenómeno real y de su estructura. A continuación se prosigue con la asociación del sistema real con su representación, es decir, el modelo.

El primer paso, el diseño del modelo, se puede ver como una *especificación* que, dependiendo de la técnica de simulación utilizada, podría ser un programa de ordenador, un modelo matemáti-

co, un formalismo lógico, o una descripción escrita. En el caso de la simulación por ordenador, el modelo es un programa de ordenador. Cuando este programa se realiza con el paradigma de agentes entonces se trata de modelos o modelado basado en agentes.

El segundo paso, una vez diseñado el modelo, es ejecutar la especificación en un ordenador. En este paso el programa de ordenador actualiza las variables de estado y de eventos del modelo a través del tiempo. Hay varias técnicas para planificar el tiempo de una simulación. Puede ser por medio de un planificador por eventos discretos, división discreta de tiempo, continuo, etc. Una vez ejecutado el modelo se obtiene un conjunto de observaciones.

Con estos datos se lleva a cabo el tercer paso de la simulación, el *análisis*. Se trata de comparar la salida con los datos recopilados durante la observación del objeto de estudio para comprobar si el modelo genera resultados consecuentes con aquellos producidos por el objeto de estudio que opera en el mundo real. Aunque esta comparación en realidad es una manera simplificada de ver el tercer paso, ya que es una tarea un poco más complicada que incluye actividades como verificación y validación del modelo y que no se especifican como requisitos en esta propuesta metodológica para la simulación por ordenador, en el siguiente capítulo se expondrá que es necesario incluirla cuando los modelos se realizan con SMA para asegurarse de la correcta interpretación e implementación del paradigma de agentes. En la Figura 4 se puede observar la lógica de la simulación por ordenador y los pasos clásicos involucrados.

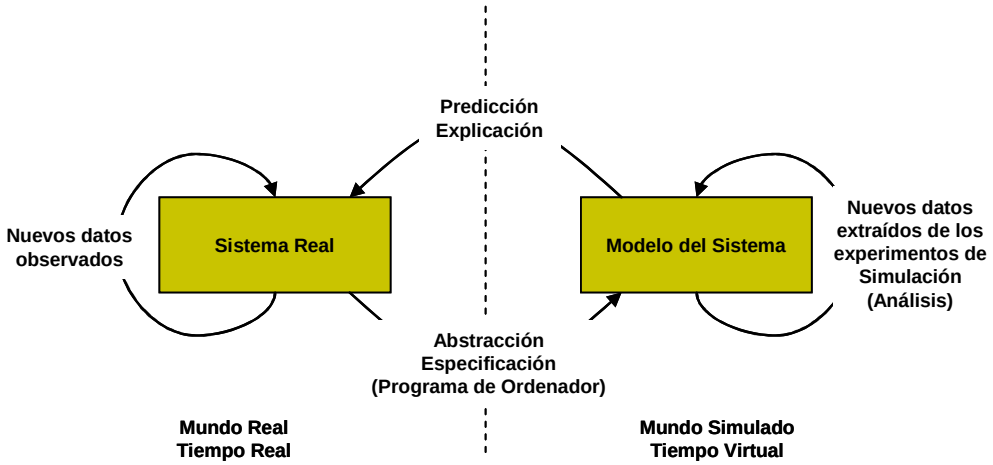

Figura 4. Relación entre el sistema real y su modelo durante la lógica de la simulación clásica

Con respecto al nivel de detalle del modelo de simulación, es decir el nivel granular de los elementos de la simulación, existen dos opciones básicas. Una es la *macro-simulación,* con una perspectiva completa del sistema. Otra la *micro-simulación,* con unidades más pequeñas con distinto estado y comportamiento (las simulaciones con SMA son una forma especial de *micro-simulación*). No hay que confundir esta última con la micro-simulación de la ciencia social, la cual

consta de unidades también pero las trata individualmente, pero no intenta modelar las interacciones de los individuos. Por otro lado, existen también formas intermedias de modelos de simulación conocidos como *multi-nivel*.

Finalmente, se ha dicho que la simulación por ordenador es una tercera forma de hacer ciencia (Axelrod 1997a), entre la teoría y la experimentación. No son experimentos en el sentido usual ya que no se manipula el objeto de estudio directamente, pero siguen la misma metodología de experimentación al permitir ejecutar la simulación muchas veces variando las condiciones iniciales y explorando los efectos de diferentes asunciones y parámetros.

2.4.3. Sociedades Artificiales como SMA

Una SA, en un sentido más particular, cuando su ingeniería se realiza a través del paradigma de agentes, es un modelo de un sistema social, como SMA, que se puede simular para su estudio. Éste construye un mundo social con agentes autónomos que imitan a *actores* reales de la sociedad, que operan en paralelo y se comunican unos con otros. La estructura social y el comportamiento del grupo emergen de la interacción local de los individuos, los cuales actúan bajo reglas de comportamiento limitadas solamente por la capacidad computacional y la información a la que tiene acceso cada agente. Así, el objetivo de las SA es el de "descubrir" los mecanismos de la emergencia de estructuras sociales macroscópicas y del comportamiento colectivo de interés.

La utilización de los SMA como modelos de simulación surge de la necesidad de representar o modelar el comportamiento humano (inteligente), la individualidad y la interacción local, la dinámica de sus interacciones, fenómenos emergentes, etc. La simulación con SMA es una propuesta de modelado "de abajo hacia arriba", que permite observar a un conjunto de individuos o agentes actuar e interactuar localmente en un entorno simulado y tiempo virtual. Los agentes cuentan con reglas de comportamiento y un estado que suele ser dinámico. El entorno depende del sistema a modelar. Puede ser espacial si los agentes están "situados" o puede ser no-espacial o continuo cuando la ubicación de los agentes es intrascendente. Las interacciones pueden ser directas entre los agentes o a través del entorno, el cual puede tener su propio comportamiento.

En cuanto a las habilidades de los agentes para tomar decisiones, es necesario considerar que en Sociología, el concepto de "*agencia*" es usado para convenir la naturaleza intencional de la actividad humana. Por lo tanto, está relacionada a conceptos como intencionalidad, libre albedrío, y el poder de lograr los propios objetivos. Cuando se aplica a agentes como programas de ordenador, el alcance del concepto de *agencia* es generalmente débil.

Esta debilidad en el concepto de agencia puede observarse especialmente en los modelos de SA actuales. El significado de *agente* en esta disciplina varía en un espectro que va desde simples objetos, pasando por agentes reactivos que cuentan con un estado interno, hasta complejos agentes

cognitivos, algunos con capacidad de aprendizaje. Es por esto que surge la discusión sobre el potencial de los SMA (como son concebidos en la Inteligencia Artificial) y su aplicabilidad y aprovechamiento en las SA. En el siguiente capítulo se revelará que la debilidad en el concepto de *agencia* es una restricción impuesta principalmente por las herramientas de desarrollo más que por cuestiones teóricas.

A continuación se examinan los aspectos metodológicos de las SA más a fondo y sus ventajas sobre otros métodos de modelado.

2.4.3.1. Aspectos Metodológicos

La primera implementación de una SA es acreditada a Schelling (Schelling 1971, 1978). Esta simulación trataba el tema sobre la segregación racial y ni siquiera se llevó a cabo en un ordenador. Sin embargo, su trabajo está considerado como uno de los clásicos en el área. En la simulación de Schelling los agentes representan personas y las interacciones de los agentes representan procesos sociales relevantes. Schelling aplicó esta simulación para estudiar los patrones de segregación durante la búsqueda de vecindario planteándose la pregunta ¿es posible obtener patrones de asentamiento altamente segregados aún si las preferencias residenciales de los individuos son pequeñas? El modelo de Schelling demostró que los *guetos* pueden desarrollarse espontáneamente, es decir, que pueden emerger patrones que no están necesariamente implicados con los objetivos de los agentes individuales, ni siquiera son consistentes con éstos. Por lo tanto, los patrones de segregación surgen de la dinámica no-lineal que genera la interacción y la interdependencia entre los agentes. Con esto pudo observar que las acciones de los individuos generan a menudo consecuencias imprevistas, procesos no intencionados por nadie.

Extendiendo la noción de Schelling acerca de crear SA a través de simulación con agentes, Epstein y Axtell crearon un modelo pionero llamado *Sugarscape* (Epstein y Axtell 1996). En numerosos experimentos computacionales de éste se observa la emergencia de agentes con una variedad de características y comportamientos, altamente sugestivos de una sociedad real, aunque algo rudimentaria. Los procesos emergentes observados incluyen muerte, enfermedad, comercio, riqueza, reproducción, cultura, etc. Por otro lado, Axelrod también construyó una SA (Axelrod 1997b), cuyo objetivo fundamental era mostrar que la convergencia local no siempre deriva en una convergencia global. Sus resultados confirman la observación generalizada de los dos ejemplos anteriores, que los micro-motivos individuales generan macro-comportamientos no intencionados.

Los ejemplos anteriores demuestran que las SA permiten la exploración de los mecanismos de la emergencia social y proporcionan nuevas perspectivas teóricas y metodológicas para su estudio. Estas SA exhiben una serie de características metodológicas que hacen de los SMA unos modelos de simulación especialmente aptos para la investigación sociológica. En general, estas características son propias de los SMA: el llamado modelado de “abajo hacia arriba”, es decir el individuo

como punto de partida, interacciones locales y paralelas, la racionalidad limitada (aunque en el caso del paradigma de agentes en la Inteligencia Artificial esta característica busca precisamente lo contrario, agentes cada vez más inteligentes), y finalmente la habilidad para capturar procesos emergentes de las interacciones de los agentes del sistema.

Las siguientes son las principales características metodológicas de las SA:

- La lógica de construcción de las SA incluye una lista detallada de agentes y sus reglas de actuación, protocolos de comunicación, etc. Son individualistas metodológicamente cuando contienen solamente representaciones explícitas de agentes individuales y de sus interacciones, y el comportamiento macro que emerge es explicado por la simulación (Conte et al. 2001) (a diferencia del individualismo metodológico de la Sociología que busca explicaciones reducidas a los individuos).
- Los agentes procesan solamente información local (nivel micro-sociológico). Así, las preferencias de los agentes, información, opciones, etc. son afectados por el comportamiento de otros agentes directamente, en lugar de estar mediados por instituciones, p. ej. el caso de los mercados en la economía. Generalmente, se asume que los agentes más cercanos tienen mayor probabilidad de interactuar o de influenciarse mutuamente que aquellos que están más separados en el entorno. Es precisamente este rasgo lo que hace de los SMA una herramienta idónea para el análisis de procesos complejos pues facilita la adaptación de los agentes (Axelrod 1997c).
- Los modelos de simulación social asumen una racionalidad limitada (Simon 1982) de los agentes. Esto implica que la información con la que cuentan los agentes no es completa, ni sus objetivos están bien definidos, ni son capaces de optimizar completamente su comportamiento (a diferencia de los agentes racionales en la economía, que asumen una racionalidad y homogeneidad completa) (Pascual 2006). Los agentes son heterogéneos en sus identidades, rasgos, preferencias, gustos o memoria; su razonamiento puede ser más o menos racional, inteligente o ingenuo; sus comportamientos pueden cambiar a lo largo del tiempo, bien porque aprendan de su experiencia pasada, bien porque aprendan de la experiencia de los demás; y pueden moverse por un entorno en busca de la satisfacción de sus preferencias, para adaptarse a él o para modificarlo (Gaylord y D'Andria 1998). Según (Axelrod 1997c), la principal alternativa a la asunción de racionalidad completa es el comportamiento adaptativo. Los SMA hacen posible la construcción de modelos con agentes con un conocimiento imperfecto (Gilbert y Doran 1994) y más realistas.
- Los modelos de simulación con SMA tienen la habilidad para capturar procesos emergentes. El propósito de las simulaciones es desarrollar agentes autónomos, como sistemas computacionales trabajando en entornos complejos, dinámicos, y con frecuencia imprede-

cibles, que sean capaces de aprender y adaptarse al mundo externo. Estos tipos de simulaciones están estrechamente vinculados al concepto de propiedad emergente, porque permiten definir modelos computacionales que sean capaces de crear un conjunto de propiedades globales de todas las características de los elementos de un sistema (Gilbert y Conte 1995). El SMA de la simulación no tiene un control centralizado, es un sistema complejo en el cual la agregación de las acciones autónomas de los agentes resulta en el comportamiento global del sistema.

Entre las ventajas que proporciona el modelado y simulación social basada en agentes con respecto a otras propuestas, hay que destacar la posibilidad de modelar una población heterogénea y dinámica, es decir individuos heterogéneos con diferentes perspectivas de su mundo social, a diferencia de otras propuestas más tradicionales que para permitir una solución matemática deben limitarse a actores homogéneos o agentes representativos. También brindan la posibilidad de modelar el entorno y la topología de las interacciones, la adaptación y evolución de los agentes reales, junto con habilidades racionales más realistas. Estos modelos también son efectivos para observar la emergencia de instituciones sociales de las acciones individuales de los agentes, y no menos importante es la capacidad para desarrollar y explorar teorías de procesos sociales.

Entre los inconvenientes, se destaca que las simulaciones de los procesos sociales complejos requieren un gran esfuerzo por parte del investigador pues requiere la estimación de un gran número de parámetros y datos sobre los cuales anclar las asunciones que realice. Paradójicamente esta dificultad es inherente al método de modelado, el cual exige un nivel de detalle de los atributos y reglas de comportamiento para cada agente, entorno, espacio, tiempo, etc. Este proceso significa un esfuerzo de sistematización tanto cualitativa como cuantitativa para decidir que atributos definir y los valores correspondientes. Esta problemática en realidad identifica un problema que siempre ha existido en la ciencia social, la disponibilidad de datos con los cuales construir modelos sólidos, sobre los cuales basar el análisis y la validación de los modelos.

2.5. Conclusiones

En este capítulo se han introducido las SA para el estudio de los sistemas sociales complejos. En su descripción se han destacado varios aspectos relevantes relacionados con la interdisciplinaridad de esta propuesta.

En primer lugar se ha tratado la complejidad de los sistemas sociales, y las sociedades humanas en particular, que repercuten directamente en su modelado y simulación. La argumentación se desarrolla en base a las características fundamentales de éstas, empezando con la misma definición de lo que es una sociedad. Una definición parcial es la de una colección de individuos aislados que

interactúan y son "complejos". Por otro lado, existe la perspectiva de las sociedades como entidades autónomas, independientes de los individuos. Este dilema en la definición es una cuestión teórica que tiene repercusiones metodológicas en el estudio de los sistemas sociales.

Específicamente, en las sociedades humanas hay que tener en cuenta las habilidades cognitivas de los individuos, notablemente "racionales", concientes de otros y de la sociedad misma. Esto sugiere modelos que capturen aspectos de la cognición humana apropiadamente, como modelos mentales, reflexivos, etc.

Otra característica central de la actividad social es la comunicación, la cual permite compartir información y llevar a cabo acciones de coordinación, que es la esencia de una sociedad. Estas mismas sociedades están arraigadas o incrustadas en un entorno, con el cual interactúan y de alguna manera las determina. La problemática surge cuando este mismo entorno incluye otras sociedades. Además es necesario tomar en cuenta la distribución en el entorno. A estas características se le puede sumar la complejidad que surge cuando los individuos interaccionan dando lugar a una dinámica no-lineal.

La complejidad de las sociedades humanas implica que los modelos pueden ser también complejos, lo que augura también una dificultad en su estudio. Se ha tomado en cuenta para describir esta complejidad las diferentes vistas alternativas de una sociedad, ya sea a nivel micro o macro, las cuales sugieren un contenido diferente para los modelos.

Se puede concluir que sin tener en cuenta estas cuestiones, el modelo de la sociedad estaría limitado para proporcionar respuestas relativas a su estudio. La complejidad que se imprima en éstos dependerá del problema de estudio y de su propósito. Según estos aspectos se podrán hacer ciertas simplificaciones. En algunos casos, modelos sencillos pueden ser suficientes para obtener resultados importantes. Por ejemplo, en el estudio etnográfico de (Axelrod 1997b) los rasgos culturales se representan con cadenas de números, y los agentes llevan a cabo acciones simples. Los resultados observados con este modelo resultaron relevantes para el estudio de la diseminación cultural. Por otro lado, se pueden requerir potentes modelos cognitivos, como el del estudio de la religiosidad de (Pavón et al. 2007a) en el cual los agentes son modelados como intencionales guiados por sus objetivos con el propósito de observar patrones sociales en las creencias de los individuos y en su toma de decisiones en una sociedad. En el estudio sobre el altruismo reproducido en (Sansores et al. 2005) también se requieren modelos cognitivos importantes para modelar otros aspectos como la dinámica de objetivos y adaptación.

Por otro lado, se analizaron brevemente las propiedades de los sistemas complejos y las teorías para su estudio. La recapitulación que hacemos sobre éstas es que uno de los principales propósitos de los estudios de la complejidad es desarrollar los conceptos, los principios y las herramientas que permitan describir las características comunes a varios sistemas complejos. Esto conduce a estudios interdisciplinarios, ya que las ideas desarrolladas para gestionar sistemas complejos en las

ciencias físicas, por ejemplo, tienen también relevancia para los sistemas en las ciencias biológicas y sociales, y viceversa.

Una vez situados en la problemática de los sistemas sociales complejos, se prosiguió con la estrategia de análisis de los sistemas sociales como *sistemas complejos adaptativos*, utilizando el modelado y simulación con SMA. Concluimos de la revisión anterior que las SA son una nueva propuesta para el estudio de sistemas sociales, una nueva forma de modelado de los procesos y fenómenos de estos sistemas.

Entre las ventajas de las SA se destaca el análisis multi-disciplinar que permiten sus modelos y las aportaciones teóricas que pueden devenir de ello. Por ejemplo, permiten modelar sistemas donde factores sociales, psicológicos y económicos son importantes y es útil considerarlos juntos.

Otra de sus ventajas es que facilitan la exploración de la emergencia social, y la observación de macro propiedades emergentes y sus efectos sobre los elementos micro del modelo, esto es, cuando las estructuras tienen efectos causales en la acción del individuo, lo cual es difícil de estudiar con modelos de ecuaciones. Así, las SA permiten explorar teorías tanto individualistas metodológicas como estructurales, teorías que intentan reconciliar la autonomía individual, por un lado, y fenómenos estructurales por otro.

Actualmente, una limitación de las SA, que restringe su relevancia para la teoría sociológica, está relacionada con la construcción de los modelos. En las SA, la descripción de los actores de la sociedad real y su comportamiento se realiza generalmente con lenguajes de programación. Por tanto, las teorías sociales que convencionalmente se han expresado en forma textual ahora requieren ser formalizadas dentro de una especificación que pueda ser ejecutada en un ordenador. Generalmente, los especialistas en esta materia se ven forzados a aprender un lenguaje de programación o aprender a programar, lo que implica un esfuerzo y costes extras que limita la rapidez de aplicación de esta estrategia y en el peor de los casos su adopción.

Por consiguiente, se puede entrever que el lenguaje para llevar a cabo la formalización de SA es fundamental. El siguiente paso será revisar las propuestas existentes para llevar a cabo esta tarea. Esta revisión nos ayudará a evidenciar la hipótesis que planteamos al inicio de este trabajo de investigación, que el tratamiento actual de la formalización de SA es insuficiente en varios aspectos

Capítulo 3.

Estado del Arte: Metodologías y Herramientas para la Simulación Basada en Agentes

Las SA se aplican como un método de investigación experimental en las ciencias sociales. Las propuestas metodológicas existentes a este respecto hacen referencia a cómo incluir este método dentro de un proceso de investigación más general, es decir, señalan en qué parte de la experimentación tiene lugar la simulación con SMA. Otras van más allá y hacen algunas recomendaciones de cómo construir modelos con SMA para su posterior ejecución en un ordenador. Este capítulo se centra en cómo se conciben e implementan las SA basándose en los métodos y herramientas utilizadas actualmente en la disciplina de la simulación social basada en agentes (SSBA). Al final se hace una evaluación de las propuestas revisadas y se señalan algunos de los aspectos para mejorarlas, los cuales son considerados en el marco metodológico que se propone en esta investigación.

3.1. Introducción

La SSBA requiere de numerosos elementos cuya integración en la construcción de un sistema no es trivial. La investigación en esta área incluye el conocimiento de varias disciplinas cómo la ingeniería del software, la simulación por ordenador y conceptos teóricos y metodológicos de las ciencias sociales. Basándose en este conocimiento se han producido metodologías y herramientas de programación para la construcción de estos sistemas.

Las metodologías que han surgido podemos clasificarlas en dos tipos, por un lado, las *metodologías de investigación* y por otro las *metodologías de desarrollo*. El surgimiento de las primeras tiene una razón lógica de ser, pues las SA como un nuevo método de investigación experimental en las ciencias sociales requieren de metodologías que orienten al usuario de cómo utilizar los modelos de simulación y sus resultados en un proceso de investigación tradicional. Este proceso demanda nuevas consideraciones debido a la naturaleza orientada a agentes del método de experimentación.

El segundo tipo de metodologías que encontramos están enfocadas a la ingeniería de los modelos de simulación. Éstas generalmente proporcionan una guía de concepción e implementación de las SA y de la ejecución de los modelos. Aunque es importante remarcar que las *metodologías de investigación* también pueden definir pasos para la construcción de los modelos, en general no hacen énfasis especial sobre éstos. En este trabajo consideramos ambas metodologías debido a que no sólo nos interesa proporcionar los métodos para la construcción de los modelos de simulación sino que también nos interesa ubicar qué partes del proceso de investigación pueden cubrirse o facilitarse mediante la propuesta metodológica que hacemos en esta tesis. En general no haremos una distinción especial entre estas metodologías y nos referiremos a ellas de forma indistinta.

Respecto a las herramientas para el desarrollo de SA, éstas también varían según el grado de implicación en el proceso de ingeniería de los modelos de simulación. Existen desde las que están orientadas lenguajes de programación, pasando por las que proporcionan entornos de desarrollo y ejecución a un nivel más abstracto, hasta las que se involucran en la especificación de los modelos a alto nivel. En general, estas herramientas son entornos de simulación que liberan al modelador de la carga de la programación de características que no son propias del dominio del problema, por ejemplo, del control de la simulación, procedimientos de entrada y salida, presentación de resultados, etc.

El análisis del estado del arte de métodos y herramientas nos ha llevado a identificar varios aspectos que esta tesis trata de mejorar:

1. *Necesidad de una estructuración del desarrollo de modelos de SSBA*. Las metodologías actuales detallan el desarrollo de las SA desde un punto de vista de la construcción clásica de modelos de simulación por ordenador, es decir, carecen de una orientación a agentes. El

concepto de agente en estas metodologías es más bien un término estándar acordado para referirse al método de simulación que un concepto con una significación metodológica. Es decir, la semántica de lo que es realmente un agente varía no solo de un modelo a otro de la misma especificación sino que también varía entre los niveles de desarrollo de un mismo modelo, por ejemplo entre el diseño y la implementación. Además, la mayoría de las sugerencias que realizan las propuestas existentes con respecto al método de desarrollo están confinadas a consejos de post-desarrollo, de validación de los modelos o se concentran en prácticas para probar simulaciones existentes (Axelrod 1997a; Gilbert 1999; Macy y Willer 2002; Gilbert y Terna 2000; Gilbert y Troitzsch 1999). Hacen falta propuestas que se concentren más en la ingeniería del software de los modelos de simulación, quizá tomando como ejemplo la ingeniería del software orientada a agentes adecuándola a los requisitos del modelado basado en agentes. Algunas propuestas preliminares se han hecho ya para una metodología para la simulación basada en agentes (Goldspink 2002; Drogoul et al. 2002). Sin embargo, éstas aún carecen del conocimiento de *cómo* y *qué* debe de realizarse en cada etapa del proceso, es decir, carecen aún de información de cómo deben construirse los modelos, y evolucionan a lo largo de las etapas del ciclo de desarrollo. Un marco metodológico que proporcione esta información y que incluya herramientas de soporte al modelado y la implementación es deseable.

2. *Necesidad de un lenguaje de alto nivel para la especificación de los modelos de simulación con SMA*. La simulación por ordenador requiere programas de ordenador y por lo tanto desarrolladores instruidos en la programación. El modelado basado en agentes permite a los modeladores describir concisamente sistemas complejos con un gran número dinámico de variables. Sin embargo, para llevar a cabo su implementación los modeladores se apoyan en métodos y lenguajes orientados a objetos o en herramientas cuya manipulación es compleja. Generalmente los modeladores no son expertos programadores, especialmente en un dominio interdisciplinario como es la SSBA. Por lo tanto, la nueva propuesta de modelado y simulación basada en agentes requiere de prácticas especiales con herramientas dedicadas a facilitar este proceso de creación, de tal forma que les permita a los expertos del dominio enfocar su esfuerzo en el modelado y evaluación de resultados y no en la implementación de los modelos. Para esto es necesario una especificación de los modelos a un nivel cercano a los conceptos que utilizaría un experto del dominio.
3. *Necesidad de un modelo computacional de agente y SMA*. Este es un requisito que deben llenar las herramientas de desarrollo de SA. En la actualidad estas herramientas carecen de un modelo de agente como los definidos en las disciplinas de la Inteligencia Artificial Distribuida o más concretamente en la de SMA, que permitan trasladar los conceptos de agentes de un nivel más abstracto de modelado a un nivel más particular de aplicación. Esto

permitiría asegurarse que en los distintos niveles de abstracción involucrados en el desarrollo se emplee el mismo vocabulario de agentes y que los agentes se utilicen no sólo a nivel conceptual sino que también la implementación se lleve a cabo realmente con agentes computacionales, cómo lo demanda la SSBA. Lo ideal es contar con un mecanismo flexible que permita adecuar este modelo según las necesidades de complejidad implícitas en los agentes, tal como un mecanismo de generación automática de los modelos computacionales.

4. *Soporte para la replicación de los modelos de simulación*. La replicación permite revisar la aparición de errores difíciles de observar en ciertas simulaciones y que pueden causar cierto sesgo en los resultados, por ejemplo, errores de implementación, errores de concepción, o por alguna característica propia de la plataforma de simulación como se demuestra en (Sansores y Pavón 2005c). La replicación de los modelos es una cuestión del proceso de investigación más que un requisito de desarrollo. Sin embargo, debido a la naturaleza de los modelos de SSBA, los cuales son en última instancia programas de ordenador, su publicación genera cierta problemática que debe ser atacada desde el marco metodológico para su estudio. La publicación de los modelos es una forma de hacerlos disponibles para que terceros puedan replicar los resultados. Idealmente, deben ser tan claros como para que un tercero pueda captar los aspectos sociales de la investigación sin ahondar en demasiados detalles. Al mismo tiempo, la información que detalla el modelo debe ser lo suficientemente explícita como para poder replicarlo. Ambos objetivos están en tensión uno con otro (Axelrod 1997a). Lo ideal es contar con un mecanismo para especificar los modelos en un nivel de abstracción más apegado al dominio del problema, de tal forma que la comprensión del modelo sea más natural para el experto y que la correspondencia de esta especificación al código fuente del modelo computacional sea clara y bien documentada. Con esto se facilita la replicación del modelo, posiblemente en diferentes plataformas.

Para evidenciar la necesidad de tratar estos aspectos llevamos a cabo una evaluación de las propuestas metodológicas que hemos encontrado en la literatura de la simulación social, por ejemplo en los trabajos de (Fishwick 1995a; Gilbert y Troitzsch 1999; Drogoul et al. 2002; Axelrod 1997a; Goldspink 2002; Moss 1998). En general, muy pocos trabajos publicados hacen referencia a las metodologías que han utilizado, y se centran más que nada en presentar los resultados de la simulación y en proporcionar descripciones textuales o gráficas del modelo, o en otros casos, del código fuente del modelo para alguna plataforma de ejecución específica. Por lo tanto, es evidente que existe poco trabajo sobre las metodologías de desarrollo. Aún en la disciplina de la simulación por ordenador encontramos trabajos que se enfocan más a cuestiones técnicas que metodológicas. Por ejemplo, en (Bratley et al. 1987) una gran preocupación de los autores es la generación de números

pseudos-aleatorios para la simulación. Aunque es verdad que estas cuestiones técnicas son necesarias para la simulación, en la actualidad las herramientas de SSBA ya se ocupan de ellas. Así, las consideraciones anteriores muestran que las plataformas para la implementación de modelos de simulación social juegan también un papel central en la SSBA pues toda la construcción y publicación de los modelos gira alrededor de estas herramientas.

Las herramientas que se han elegido en este trabajo de investigación son aquellas en principio más adecuadas para el modelado de sistemas sociales por las facilidades que proporcionan para adaptarse al modelado de procesos sociales. En la actualidad estas herramientas no están basadas en teorías sociales específicas, algunas permiten ser adaptadas más fácilmente que otras a las necesidades de modelado del usuario (en el caso de que el usuario desee implantar una teoría específica), por lo cual este es uno de los criterios de selección que se ha utilizado. Además, se consideraron las herramientas más referenciadas en el dominio de la simulación social. Aunque no existe una herramienta que satisfaga todas las necesidades del modelado social, se han considerado las que simplifican el proceso de desarrollo y las que son de código abierto susceptibles a ser adaptadas.

Teniendo en cuenta estos criterios clasificamos las herramientas en tres tipos: (1) librerías de programación como Repast (Collier et al. 2003), Mason (Luke et al. 2003) y Swarm (Minar et al. 1996), (2) herramientas de desarrollo rápido de aplicaciones como NetLogo (Wilensky 1999), y (3) herramientas de modelado gráfico como SDML (Moss et al. 1998) y Sesam (Kluegl 2001).

El análisis de las metodologías y las herramientas de desarrollo ha tenido en cuenta varios aspectos:

1. *La orientación a agentes de las metodologías, sus actividades y las facilidades para su aplicación*. En cuanto a las metodologías de desarrollo, prácticamente nos centramos en estudiar cuales son sus propuestas metodológicas para la creación de modelos basados en SMA, cuáles son las actividades en el proceso de desarrollo que proponen, cómo realizarlas y con qué herramientas.
2. *Facilidades de modelado*. Una de las cuestiones más importantes que buscamos en las herramientas de implementación de sistemas de SSBA son sus facilidades de modelado de alto nivel. Una de estas facilidades puede ser algún lenguaje de especificación que reduzca el conocimiento de programación requerido por los modeladores y que incremente su concentración en los aspectos teóricos del modelo. Este lenguaje debe corresponderse con la arquitectura o el modelo de agentes a nivel de implementación. Otras facilidades pueden ser abstracciones del dominio listas para utilizarse sin necesidad de programación y abstracciones organizacionales que son componentes del SMA y que estructuran explícitamente una organización como roles y grupos.

3. *Arquitectura interna de los agentes*. La arquitectura interna de los agentes que promueven las herramientas de implementación se refiere primordialmente a la complejidad de la racionalidad que podemos incluir en la implementación de los agentes, por ejemplo, en forma de patrones predefinidos genéricos que vayan desde agentes reactivos a intencionales. Si se diera el caso que la herramienta proporcione un lenguaje de especificación lo ideal sería que hubiera una correspondencia directa entre los conceptos de racionalidad que se especifican y los que se implementa con esta arquitectura.
4. *Facilidades para la comunicación de los modelos*. Cuáles son los medios que se utilizan en estas herramientas para la comunicación de los modelos entre diferentes usuarios, modeladores o terceros. Éstos varían desde código fuente a diagramas tipo UML.
5. *Facilidades para el desarrollo.* En este punto estudiamos cuáles son las facilidades que proporciona cada herramienta para facilitar la implementación, por ejemplo, una interfaz gráfica, librería de funciones, diferentes entornos espaciales, etc. La facilidad de uso la evaluamos en relación al nivel de conocimiento de programación requerido para su utilización: mientras más abstracciones se proporcionen a través de una interfaz gráfica, más fácil debería ser el desarrollo; mientras más conocimiento de programación se requiera, más complicado puede resultar para un programador sin suficiente experiencia.
6. *Facilidades para la experimentación y análisis.* El soporte para la experimentación se refiere al almacenamiento y control de los datos arrojados por una serie de ejecuciones del modelo para ser analizados posteriormente. En este caso también se refiere a la posibilidad de observar ciertos eventos durante la simulación, por ejemplo por medio de ventanas de observación del estado interno de los agentes, es decir, su estado mental y el efecto de ciertas acciones e interacciones. También podrían observarse eventos a nivel del modelo o del sistema, por ejemplo, la creación de algún agente. Otra forma importante de experimentación sería la posibilidad de modificar dinámicamente las condiciones de la simulación, por ejemplo la asignación de variables, la selección de acciones de los agentes, su estado interno, etc. Finalmente, el análisis se refiere a herramientas para la evaluación de los datos, como pueden ser paquetes gráficos o estadísticos.

En la línea de (Drogoul et al. 2002) consideramos que la ingeniería de sistemas de SSBA requiere extender los métodos usados tradicionalmente en la simulación social. También consideramos que las herramientas requieren ciertos mecanismos más orientados al paradigma de SMA y una visión más integradora para conciliar la perspectiva interdisciplinaria de las áreas involucradas en la SSBA. En este capítulo nos encargamos de evidenciar esta problemática y de delinear las características deseables en una posible solución.

El resto del capítulo se estructura como sigue. La sección 3.2 hace una revisión de las propuestas metodológicas encontradas en algunos trabajos de la simulación social para la construcción de SA. Principalmente se analiza la orientación de estos métodos a la construcción de modelos basados SMA. La sección 3.3 presenta un estudio de las herramientas más extendidas para la construcción de SA. La sección 3.4 revisa los patrones de diseño que las herramientas evaluadas proponen para el desarrollo de SA. Finalmente, la sección 3.4 hace una síntesis de la evaluación de los métodos y herramientas vistas en este estado del arte y de las características que asume la propuesta presentada en esta tesis.

3.2. Metodologías

En esta sección se presentan las principales propuestas metodológicas para la construcción de SA extraídas de la literatura actual en la disciplina de la SSBA. Este recorrido aborda propuestas que han nacido en el dominio de la simulación social con el propósito de orientar a los expertos de las ciencias sociales para llevar a cabo una investigación experimental con modelos de simulación basados en agentes.

En el capítulo anterior estudiamos la lógica de la simulación por ordenador y los pasos genéricos o tareas que se proponen habitualmente para ésta. Básicamente son tres, tal como se puede observar en la Figura 5.

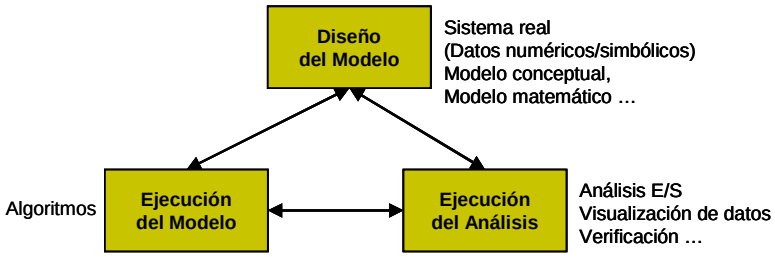

Figura 5. Tareas en el proceso de simulación por ordenador (Fishwick 1995b)

Para su aplicación a la SSBA, este planteamiento (Fishwick 1995b) es demasiado genérico y así se puede ver que en algunas tareas no hay una clara separación de funciones. Por ejemplo, en el *diseño del modelo* no se define cómo se lleva a cabo la traducción del modelo inicial a un modelo computacional. En principio, se describe como primer paso la adquisición de conocimiento para la creación de un modelo formal, ya sea matemático, conceptual, etc. Posteriormente este modelo será traducido a algún algoritmo para su ejecución. Sin embargo, no queda claro cómo se lleva a cabo la especificación y la construcción del modelo computacional, o cómo se pasa del diseño a la implementación.

En base a la recopilación de pasos sugeridos en diferentes trabajos de simulación por ordenador, (Ramanath y Gilbert 2003) presentan una síntesis más detallada de los pasos que generalmente suele cubrir el proceso de investigación basado en la simulación por ordenador, tal como se muestra en la Figura 6. Los pasos que habitualmente abarcan las propuestas metodológicas actuales son la *implementación*, la *simulación*, el *análisis*, y la *publicación de resultados*.

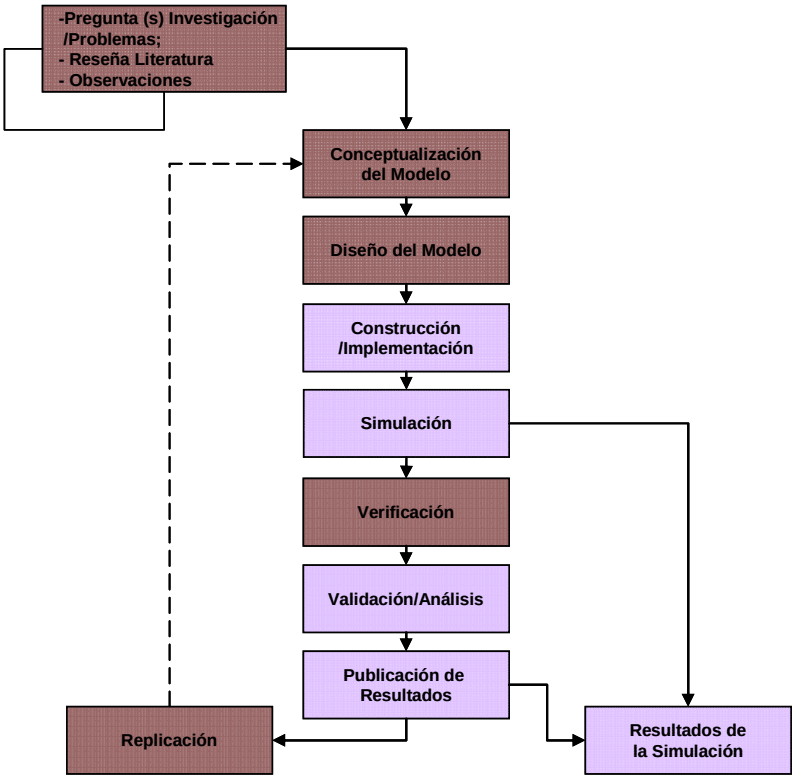

Figura 6. Pasos genéricos en el proceso de investigación basado en la simulación (versión adaptada de (Ramanath y Gilbert 2003))

Esta propuesta es un marco de investigación genérico en cuanto a que no hace mención a qué estudios de simulación está orientado, con lo cual cada metodología tendría que especificar a qué tipo de modelos se refiere cuando hace mención a alguno de estos pasos, por ejemplo, modelos con agentes en el caso de la SSBA. Diferentes autores proporcionan versiones reducidas del marco propuesto en la Figura 6. Sin embargo, el grado hasta el cual estos pasos son cubiertos depende del tema y de los objetivos del estudio de simulación.

Este marco nos servirá de referencia para la evaluación de los métodos propuestos en los trabajos de (Goldspink 2002; Axelrod 1997a; Gilbert y Troitzsch 1999; Drogoul et al. 2002) que se comentan a continuación.

3.2.1. Gilbert y Troitzsch

Según (Gilbert y Troitzsch 1999; Troitzsch 1997; Gilbert 1999) el proceso de investigación basado en la simulación por ordenador incluye las siguientes tareas: *conceptualización del modelo*, *diseño del modelo*, *construcción del modelo*, *simulación*, *verificación y validación*, *análisis* y *publicación*. La Figura 7 muestra la secuencia de estos pasos; que se explican a continuación:

- *Conceptualización del modelo*. Consiste en definir el objeto de estudio y el alcance del problema a ser modelado. En este paso también se incluyen las observaciones que deben realizarse del objeto de estudio con el fin de proporcionar los parámetros y las condiciones iniciales requeridas por el modelo. En base a estas observaciones se pueden hacer asunciones para diseñar el modelo.
- *Diseño del modelo*. En este paso se plantea el dilema al que se enfrenta el diseñador a la hora de decidir qué incluir y qué excluir en el modelo. El resultado será un modelo abstracto que incluya el mínimo de asunciones y que a la vez sea aplicable tanto como sea posible a diversas circunstancias.
- *Construcción del modelo*. El siguiente paso, una vez diseñado el modelo, es su construcción, es decir, su traducción a un programa de ordenador. Aquí los autores reconocen que ningún lenguaje de programación puede proporcionarle al diseñador todos los prerrequisitos para la simulación, como son: un lenguaje compilado, bien estructurado, eficiente, que permita un refinamiento incremental y depuración fácil y rápida, con librerías gráficas, y que sea familiar para el modelador y terceros, con tal de posibilitar la replicación de los resultados. Para esto también es deseable que el lenguaje sea portable.
- *Verificación*. El siguiente paso, cuando ya se cuenta con un modelo ejecutable y se puede llevar a cabo la *simulación,* es verificar que el programa realice lo que uno espera. Se verifica la correcta transformación de la representación abstracta o modelo conceptual al código del programa o modelo de simulación. Prácticamente, este es un paso de depuración, aunque puede ser muy difícil de llevar a cabo con simulaciones complejas. Además, la mayoría de las simulaciones sociales dependen de números seudo-aleatorios para simular los efectos de variables inmensurables y efectos aleatorios, así que ejecuciones repetidas suelen producir salidas diferentes.
- *Validación*. A este paso le concierne validar que la simulación es un buen modelo del objeto de estudio. Un modelo cuyo comportamiento corresponda al comportamiento del objeto de estudio es considerado "válido". Para hacer esta prueba se propone comparar los datos observados con los datos arrojados por la simulación, es decir los datos simulados. Si existe lo que los autores llaman *similitud estructural*, esto es, que haya una co-variación en circunstancias similares, entonces esto cuenta como una evidencia de la adecuación del mode-

lo como una representación del objeto de estudio. Además, (Troitzsch 1997) arguye que una prueba que podría realizarse para verificar la *validez* de las simulaciones sería experimentar su habilidad para explicar observaciones pasadas, sobre todo aquellas simulaciones que serán usadas para predecir.

- *Análisis*. Se sugiere un análisis de sensibilidad una vez que el modelo es considerado válido, es decir, qué tan sensible es el modelo a pequeños cambios en los parámetros y condiciones iniciales. Se trata de analizar hasta qué punto el comportamiento de la simulación es sensible a las asunciones hechas durante el diseño.
- *Publicación*. Este último paso se refiere a la publicación de los resultados de la investigación, colocándolos en un formato listo para su uso por terceros.

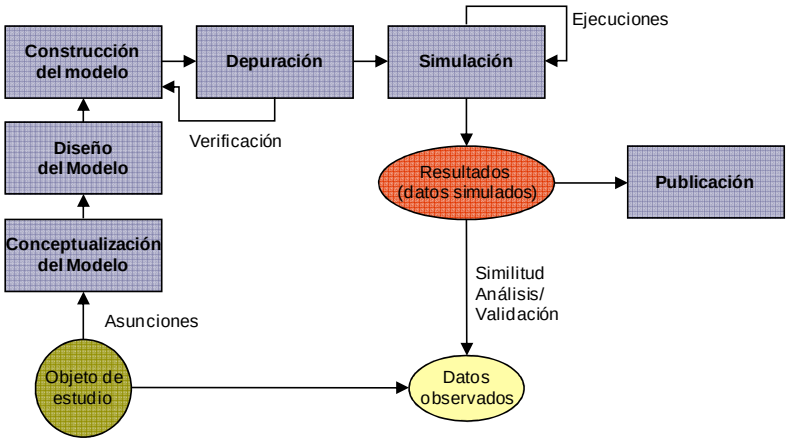

Figura 7. Propuesta metodológica para la investigación con la SSBA de Gilbert y Troitzsch (diagrama sintetizado de (Gilbert y Troitzsch 1999)).

Esta propuesta está mejor estructurada que la descrita al inicio de esta sección, ya que considera claramente los aspectos desde la observación del fenómeno de estudio hasta la validación final del comportamiento del modelo con la del sistema real. Sin embargo, debido a su carácter general presenta ciertas limitaciones en cuanto al diseño e implementación de modelos de simulación basados en agentes. En el paso del *diseño del modelo* no se hace referencia al tipo de modelos que se obtienen o qué modelos deben generarse y mucho menos hace referencia a algún lenguaje en particular con que llevar a cabo el diseño. Este paso se refiere principalmente a la abstracción del fenómeno a estudiar, en un modelo formal, aunque también puede ser un modelo no formal, como una descripción textual. Es evidente que la transformación del modelo inicial a un programa de ordenador no es trivial, sin embargo, no se detallan los pasos entre estas dos representaciones durante la *construcción del modelo*. Este último paso tampoco hace referencia a cómo llevar a cabo la implementación del SMA ni la arquitectura interna de los agentes ni sus interacciones.

3.2.2. Axelrod

En uno de las trabajos más referenciados en la literatura de la SSBA, (Axelrod 1997a) destaca tres aspectos del proceso de investigación basado en la simulación que tienen lugar una vez desarrollado el *modelo conceptual*. Estos son: *programación del modelo*, *análisis de los datos*, y *compartir resultados*. Aunque pareciera una propuesta simplificada en realidad estos pasos contienen otros aspectos a considerar y que forman parte también del marco genérico que presentamos anteriormente. Las características más relevantes de esta propuesta son:

- *Programación del modelo*. Este paso corresponde a la *construcción del modelo* y para ello se recomiendan algunos lenguajes de programación como Java, o Visual Basic para programadores menos experimentados. También se especifican los objetivos que debe alcanzar este paso: *validez*, *utilizable*, *extensible*. La *validez* se refiere a que el programa implemente correctamente el modelo, es decir, que los resultados inesperados no sean un reflejo de errores en la programación sino una consecuencia del modelo mismo. A esta clase de validez se le conoce como "*validez interna*" y corresponde al paso de la *verificación* que se explicó anteriormente. La "*validez externa*" (o *validación*) no se considera en esta propuesta. Que sea *utilizable* se refiere a que permita a terceros que ejecuten el programa interpretar su salida, y comprender cómo funciona. Que sea *extensible* significa que permita a futuros usuarios adaptar el programa a nuevos usos, por ejemplo que otros investigadores puedan modificar el programa para probar una nueva variante del modelo.
- *Analizar los resultados*. Al igual que (Gilbert y Troitzsch 1999), Axelrod también subraya que el análisis es difícil debido a que múltiples ejecuciones del mismo modelo pueden diferir debido a diferencias en las condiciones iniciales y a eventos estocásticos. Remarca que los resultados son *dependientes del recorrido*, es decir que para comprender los resultados es necesario comprender el detalle histórico de una ejecución dada. Esta descripción puede llevarse a cabo como una especie de descripción cronológica, desde el punto de vista de un solo actor o desde un punto de vista global.
- *Compartir resultados*. El método principal señalado para compartir los resultados es mediante la publicación. Axelrod menciona algunos problemas que presentan las simulaciones en las ciencias sociales, entre estos: los resultados de la simulación son sensibles a los detalles del modelo, por lo cual, a menos que el modelo sea descrito en detalle, el lector será incapaz de replicar o aún más, comprender completamente lo que se hizo en la simulación. El análisis de los resultados con frecuencia incluye descripciones narrativas. Para mostrar cómo se han hecho inferencias del estudio de estas descripciones usualmente se requiere bastante espacio, a diferencia de un análisis estadístico cuyos resultados se pueden describir de forma más breve.

Según Axelrod, los tres pasos anteriores se realizan virtualmente en todos los modelos publicados. Sin embargo, *la replicación* es un cuarto paso en el proceso de investigación que propone como necesario y que casi nunca se realiza. Principalmente destaca que la *replicación* es necesaria, como en cualquier otro tipo de investigación experimental, para confirmar la robustez de los resultados publicados, ya que permite validar el modelo en su dominio de investigación. En este trabajo y en (Axtell et al. 1996) se presentan los principios de la *replicación* como sigue:

- *Replicación*. Para la replicación sugiere la *alineación de los modelos computacionales*, es decir, una comparación minuciosa del modelo a replicar y el modelo nuevo o modificado para verificar si producen resultados equivalentes bajo condiciones equivalentes. Generalmente la *replicación* implica la re-implementación del modelo computacional original. Se distinguen tres niveles de *replicación*: 1) *Identidad numérica*, en el cual los resultados son reproducidos exactamente, aunque es importante recordar que este nivel solo se puede alcanzar si se utiliza el mismo generador de números aleatorios y la misma semilla. 2) *Distribución equivalente*, se obtiene cuando la distribución de los resultados no puede distinguirse estadísticamente. Este nivel es considerado suficiente para la mayoría de los casos. 3) *Equivalencia relacional*, en el cual dos modelos tienen la misma relación interna entre sus resultados, por ejemplo, que en ambos modelos una variable particular sea una función cuadrática del tiempo. Este es el nivel más débil, sin embargo debido a que la mayoría de los resultados importantes de las simulaciones sociales son cualitativos más que cuantitativos, la *equivalencia relacional* es algunas veces un estándar de *replicación* suficiente.

En la Figura 8 presentamos nuestra concepción de la secuencia de pasos metodológicos propuestos por Axelrod. Como último punto sugerido en esta propuesta se menciona que una documentación completa de este proceso de investigación debe incluir el código fuente para ejecutar el modelo, una descripción completa del modelo, otra descripción de como ejecutar el programa, y de cómo comprender los resultados.

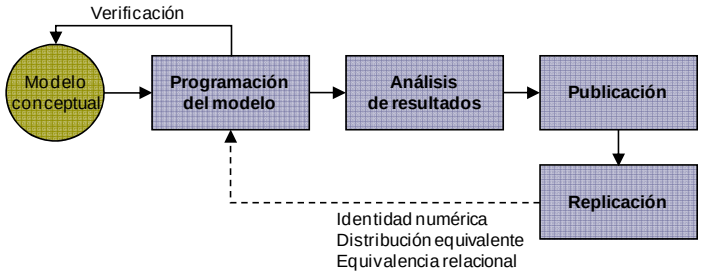

Figura 8. Propuesta metodológica para la investigación con la SSBA de Axelrod (diagrama sintetizado de (Axelrod 1997a)).

Esta propuesta está enfocada principalmente a sugerencias post-desarrollo de las simulaciones, como la *validación* y *publicación de resultados*. Se distingue por considerar la *replicación* de los modelos como una de las etapas necesarias en el proceso de investigación, además de sugerir los niveles de replicación posibles. Sin embargo, empieza dando por hecho que se cuenta con un modelo conceptual, aunque no se menciona de qué tipo, ni formalismo del modelo. El primer paso se refiere directamente a la programación del modelo, sin mayor especificación de cómo pasar de este modelo conceptual a la *construcción del modelo*. No hace ninguna consideración respecto al diseño. En cuanto a la implementación tampoco hace consideraciones especiales si tomamos en cuenta que el sistema a desarrollar es un SMA.

3.2.3. Goldspink

En (Goldspink 2002) se hace una propuesta de cómo llevar a cabo el proceso de investigación basado en la SSBA. El trabajo de Goldspink se destaca por considerar en su propuesta el diseño de los *agentes* que conforman el sistema, aunque no considera realmente las implicaciones del diseño, sino que se conforma con mencionar cómo podría analizarse un sistema para determinar qué agentes son necesarios. En general presenta tres grandes tareas que comprenden el *diseño de la simulación*, *diseño de los agentes* y el *análisis de resultados*. Estas dos últimas quedan comprendidas dentro de la primera. La propuesta se sustenta en otras metodologías para facilitar cada una de estas tareas:

- *Diseño de la simulación*. Para diseñar la simulación Goldspink propone utilizar la metodología de (Fishwick 1995b) y los pasos: *diseño del modelo*, *ejecución del modelo*, y *análisis del modelo*, los cuales requieren de habilidades diferentes. Brevemente, el *diseño del modelo* implica desarrollar un formalismo abstracto que capture las características de interés de la teoría o de una situación real del mundo. La *ejecución del modelo* implica transferir el diseño a una plataforma de simulación adecuada, codificándolo en un lenguaje de programación apropiado. Por último el *análisis* depende del nivel de abstracción que se haya considerado en el diseño, y este nivel de abstracción depende a su vez de las respuestas que se esperan obtener del modelo. Por lo tanto, diferentes niveles de análisis pueden ser necesarios, aunque no se hace énfasis en ninguna técnica en particular.

  Goldspink también se vale de la propuesta de (Axelrod 1997a) para requerir que el diseño y la implementación de la simulación busque cumplir los tres objetivos sugeridos por Axelrod: *validez interna*, *utilizable* y *extensible*. Un punto que enfatiza Goldspink es el de la *verificación* (*validación interna*). Según éste, un gran problema para establecer la *verificación* de los modelos es la *emergencia*. Un fenómeno emergente es contra intuitivo, y los resulta-

dos han de demostrarse ser resultados del modelo y no artefactos o errores de la codificación, de la plataforma o del método utilizado.

- *Diseño de los agentes*. Goldspink hace especial énfasis en este paso, donde se especifican las características de los agentes y sus acciones, así como su capacidad de interactuar. Toma la propuesta de (Parunak et al. 1998) para analizar el sistema y determinar qué agentes son apropiados y sus posibles acciones. Esto es, se descompone la descripción narrativa del modelo, teniendo en cuenta que los sustantivos indican los agentes potenciales del sistema y los verbos las acciones que necesitan llevar a cabo, y con esto último determinar también el tipo o la clase de agente. Un problema potencial que señala es el de la *reificación excesiva* en un modelo, pues puede causar problemas como incluir asunciones inapropiadas que presentarán dificultades para su implementación y validación futura.
- *Análisis de resultados*. En cuanto al análisis no hace ninguna propuesta en particular, sugiere simplemente basarse en la propuesta de (Axelrod 1997a) explicada anteriormente.

Por último, en cuanto a la *validación* es importante la observación que hace Goldspink, en el sentido de que si la simulación se usa para comprender el sistema que se ha modelado, entonces es difícil validar directamente el modelo con el sistema real, porque aún no se comprende completamente (por eso mismo se ha realizado la simulación). Además si el sistema modelado es complejo, normalmente no será posible hacer una comparación completa entre el comportamiento del modelo y el real. En cuanto a la *replicación* de la simulación en distintas plataformas, menciona que es una forma de validar que la divergencia del comportamiento del modelo y del sistema real no sea por cuestiones de artefactos de la plataforma de simulación. En la Figura 9 se muestra la síntesis de la propuesta metodológica de Goldspink.

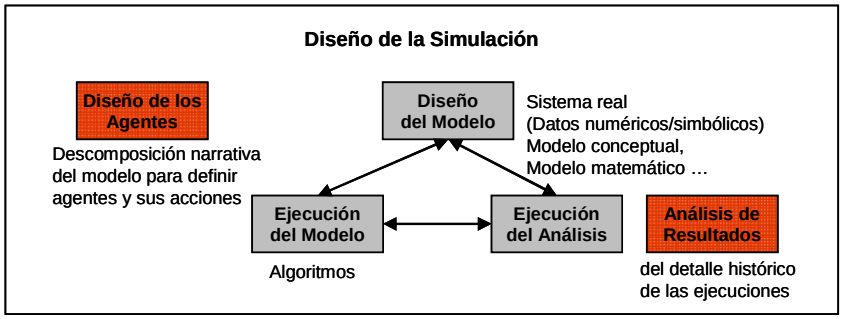

Figura 9. Propuesta metodológica para el desarrollo de sistemas de SSBA de Goldspink (diagrama sintetizado de (Goldspink 2002)).

La propuesta de Goldspink hace reflexiones interesantes con respecto a la *validación*. Aprovecha otras propuestas para formar un marco metodológico más completo para la SSBA. Es una de las primeras propuestas en considerar la característica particular de los modelos de SSBA, esto es,

que están constituidos por *agentes*, y que es necesario considerar su diseño. Sin embargo, la etapa de *diseño de los agentes* que propone no explicita cómo llevar a cabo este proceso, qué lenguajes podrían utilizarse y qué arquitecturas de agentes pueden ser útiles para llevar a cabo esta tarea. Además es difícil poner en práctica esta metodología pues no explica cómo integrar el diseño de los agentes con el del *modelo*, ni cómo llevar a cabo la implementación de los agentes.

3.2.4. Drougol et al.

Esta propuesta metodológica(Drogoul et al. 2002) es una de las más orientadas al diseño de simulaciones con SMA, a diferencia de las anteriores que se centran simplemente en la simulación por ordenador. Además es una propuesta metodológica de desarrollo que hace especial énfasis en los conceptos de agentes en los diferentes niveles de diseño que propone. Se centra específicamente en el *modelado* basado en roles, en el cual lo más interesante es *qué* es producido por *quién* en el ciclo de vida de la simulación. Para producir una simulación se propone seguir un proceso de diseño formado por tres participantes que desempeñan diferentes roles, aunque esta correspondencia no es fija, ya que un mismo participante puede desempeñar dos o los tres roles propuestos. No obstante, esto no es recomendable pues cada uno aporta un conocimiento diferente. Los tres roles y los modelos que produce cada uno se detallan a continuación:

- *Rol temático*. Involucra a los expertos de algún dominio en particular o en un tema específico que buscan explicaciones a sus teorías o buscan comprender algún fenómeno específico. Este actor tiene conocimiento del objeto de estudio por medio de observaciones que ha llevado a cabo, realiza teorías y asunciones sobre el mismo y se hace preguntas que busca resolver por medio de la simulación. El conocimiento de este actor se define en dos niveles, el *macro conocimiento* que es el conocimiento global que se tiene del sistema objeto de estudio y el *micro conocimiento* que es el conocimiento local que se tiene de los *individuos*. El propósito de este rol es producir el *modelo del dominio*, que contiene "agentes" y comportamiento de las observaciones que ha realizado y se define usando algún lenguaje específico del dominio.
- *Rol modelador*. Tiene como objetivo traducir el *modelo del dominio* a algo más formal que pueda ser implementado por un programador. Este rol produce *el modelo de diseño* para lo cual define *agentes conceptuales* que se han interpretado del *modelo del dominio*. Este modelo es a la implementación basada en agentes lo que un diagrama UML es a un programa orientado a objetos. Se sugiere que los conceptos utilizados en este modelo se tomen de los SMA como por ejemplo (Jennings 2000): interacciones, comportamiento, comunicaciones, tipo de entorno, etc. El objetivo de este rol se logra mediante una iteración entre este nivel y el nivel del *rol temático* hasta completar el diseño.

- *Rol programador*. Su objetivo es escribir un programa de ordenador proponiendo un modelo que permita la comunicación con el *modelador*. El modelo que produce se conoce como *modelo operacional*, y debería definir agentes computacionales por medio de alguna técnica de implementación (p.ej. objetos). Este modelo en realidad permite que los otros actores puedan comprender y cambiar lo que se va a implementar. Utiliza conceptos técnicos como distribución de agentes, técnicas de planificación de tiempo, etc.

El *rol programador*, una vez que cuenta con el *modelo operacional*, puede proseguir con la *construcción del sistema computacional* para ejecutarse en lo que se conoce como el *entorno simulado*. En la Figura 10 podemos observar los modelos producidos por estos roles en el proceso de diseño de esta propuesta metodológica.

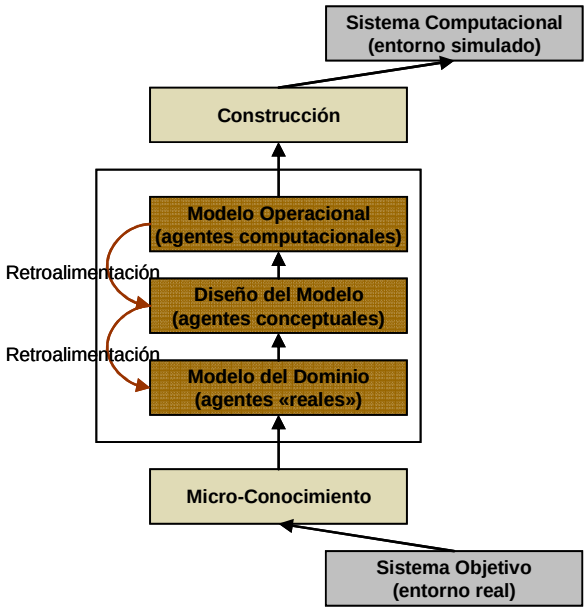

Figura 10. Propuesta metodológica para el desarrollo de simulaciones basadas en SMA de (Drogoul et al. 2002).

Esta propuesta tiene varias ventajas sobre las propuestas revisadas anteriormente en lo que al desarrollo respecta. Propone ciertas prácticas de la ingeniería del software clásica para el desarrollo de simulaciones, principalmente incluye el *diseño* y realiza un desacoplamiento del modelo conceptual y la implementación a través de este modelo. También menciona la idea de mantener el concepto de agente en los tres niveles de modelado, y de realizar la implementación con alguna tecnología de SMA que garantice la utilización del paradigma hasta la ejecución de la simulación. La propuesta de los roles es también clave en esta metodología, ya que permite agilizar el desarrollo pues cada experto puede gestionar conceptos apropiados a su conocimiento y explotar mejor los recursos. Por ejemplo, los lenguajes de programación por el experto programador.

Sin embargo, en lo que respecta a las necesidades que planteamos en este capítulo con respecto a las metodologías de desarrollo, esta propuesta no define instrucciones acerca de *cómo* describir cada uno de los aspectos considerados. En cuanto al diseño no se propone lenguaje de modelado de agentes, ni la arquitectura de agentes, ni los diseños que se han de elaborar (por ejemplo, dependiendo del lenguaje podría darse el caso de requerir un modelo para cada agente del sistema, para el entorno, etc.). En la implementación tampoco se considera cómo asegurarse de la realización computacional de los agentes y de su verificación.

La metodología está enfocada a servir como guía para ubicar *cuándo* y *qué partes* ha de desarrollar cada experto en el desarrollo de simulaciones con SMA. No obstante, el detalle de la definición de *qué* elaborar y *cómo* aún queda vaga en esta propuesta. Una metodología para la ingeniería del software tradicional generalmente viene acompañada no solo de recomendaciones y prácticas, sino también de herramientas que facilitan el desarrollo. Un proceso de desarrollo software especializado en agentes queda incompleto sin herramientas tanto de diseño o especificación formal como de implementación de agentes. Sobre todo es importante considerar que esta metodología es aplicada en un dominio distinto a la Informática, en este caso las ciencias sociales, por lo tanto, mientras menos decisiones de diseño e implementación tenga que tomar el usuario final, mucho mejor.

3.3. Lenguajes y Herramientas

En este apartado se presenta un estudio de los lenguajes y herramientas que facilitan el proceso de desarrollo de la simulación social basada en SMA. Como hemos visto anteriormente, este proceso esta compuesto de una serie de etapas en las que se llevan a cabo diferentes actividades que generan diferentes artefactos, ya sean éstos de diseño, de implementación, o de análisis. Además, estas actividades son realizadas, en principio, por diferentes actores con diferentes aptitudes y conocimientos. Por lo tanto, uno de los retos no sólo de las metodologías sino también de las herramientas para la SSBA es entrelazar las diferentes terminologías y puntos de vista de estos actores durante la realización de sus actividades.

Para esto, existen en el dominio de la simulación social varias herramientas que ayudan a realizar las actividades del proceso de desarrollo y que al mismo tiempo intentan entrelazar de forma coherente estas actividades. El grado de alcance de ambos objetivos determina el tipo y orientación de las herramientas. Esto es, mientras que algunas herramientas se orientan más a ayudar a realizar una actividad en particular dentro del marco metodológico, otras abarcan más de una actividad de forma más integradora. Son estas últimas las que revisamos en esta sección.

Entre la categoría de herramientas que soportan varias actividades también podemos distinguir dos tipos según el nivel de conocimiento de programación requerido para su utilización. Estos son, las librerías de programación y ejecución de simulaciones, y los entornos de desarrollo rápido de aplicaciones. Así, podemos encontrar herramientas que soportan las mismas actividades, por ejemplo la implementación y el análisis de los modelos, en forma de librerías de programación o en forma de entornos de desarrollo rápido de aplicaciones. Las primeras suelen ser más flexibles pero se requiere mayor conocimiento de programación. Las segundas son más fáciles de usar pero más restrictivas en su aplicación.

A continuación revisaremos tres tipos de herramientas que caen en la categoría anteriormente descrita. Primero revisaremos las librerías de programación para la simulación como Repast (Collier et al. 2003), Mason (Luke et al. 2003) y Swarm (Minar et al. 1996). Después revisaremos los entornos de desarrollo rápido de aplicaciones como NetLogo (Wilensky 1999). Y por último los entornos de desarrollo rápido de alto nivel como SDML (Moss et al. 1998) y Sesam (Kluegl 2001). La función de estas herramientas es la de permitir la creación de laboratorios artificiales a usuarios que son ocasionalmente especialistas en informática. Por lo tanto lo ideal es que suministren facilidades como: un entorno de diseño para la simulación, un lenguaje de especificación de alto nivel, un modelo de SMA flexible para el diseño e implementación de la simulación, abstracciones del dominio, facilidades para la ejecución y el análisis, mecanismos para la transformación automática del diseño a la implementación, abstracciones de la infraestructura del sistema de simulación (e.g. planificador, entornos, etc.), así como abstracciones organizacionales como grupos y roles. En todas las herramientas revisadas exploraremos con cuales de estas facilidades cuentan y a qué actividades del desarrollo están enfocadas.

3.3.1. Swarm

La herramienta Swarm (Minar et al. 1996), desarrollada por el *Instituto Santa Fe* (http://www.santafe.edu), es la primera herramienta dedicada al modelado y simulación basado en agentes y una de las más conocidas para la construcción de este tipo de modelos. Su objetivo es el de proporcionar un conjunto estandarizado de herramientas software que pueda usarse en una gran variedad de sistemas. Más que un lenguaje orientado a la programación de simulaciones con agentes, Swarm es un sistema y un marco de programación orientado a objetos. Tanto el conjunto de librerías que proporciona, como el sistema mismo están implementados en Objective C, una extensión orientada a objetos del lenguaje C. Aunque posteriormente se ha proporcionado una interfaz Java (Burkhart et al. 2000), por lo que las simulaciones pueden escribirse en ambos lenguajes.

Un *modelo* en Swarm es una colección de agentes independientes en un entorno, interactuando por medio de eventos discretos. Su dinámica se especifica como un *programa* o una lista de even-

tos discretos que ocurren en los agentes y en el entorno a través del tiempo. Las acciones de los individuos tienen lugar en momentos específicos en el tiempo, y el tiempo avanza solamente por eventos. Así, el *programa* es una estructura de datos que combina acciones con el momento en que éstas deben de ocurrir.

Los *agentes* en Swarm son modelados directamente como objetos. Los tipos de agentes son clases, y los agentes específicos son objetos, instancias de una clase dada. Los métodos de los objetos implementan las acciones del agente que serán ejecutadas como una reacción a eventos. El *programa* de un modelo basado en agentes es por lo tanto un conjunto parcialmente ordenado de acciones para ser ejecutadas por los objetos.

Una parte de las librerías de Swarm está dedicada a la infraestructura fundamental para la simulación. Estas librerías implementan objetos de simulación genéricos y estructuras de datos, entre éstas se incluye una clase abstracta de agente. Las librerías con patrones de agentes y funciones predefinidas son facilidades para el desarrollo que pueden concretarse para construir un modelo de agentes de forma más fácil.

La arquitectura de Swarm está dividida en dos niveles, por un lado se encuentra el nivel del *Modelo Swarm* y por otro el nivel del *Observador Swarm*. De esta forma se desacopla la lógica de la simulación de su representación gráfica, es decir, las acciones de ejecución de las acciones de observación. A través de esta arquitectura se obtienen las facilidades para la experimentación y el análisis. Además Swarm proporciona interfaces que permiten observar el estado interno de los agentes y las variables.

Asimismo, Swarm proporciona facilidades de modelado, con abstracciones de la infraestructura de simulación. Estas librerías ayudan a realizar tareas de programación comunes por medio de la implementación de algoritmos básicos y estructuras de datos estándar, por ejemplo abstracciones para el *planificador* de la simulación, algoritmos para la generación de números seudo-aleatorios, etc. Las abstracciones y estructura de éstas que ha definido Swarm se consideran *de facto* estándar y se han adoptado por la mayoría de las herramientas de la simulación social posteriores. Por otro lado, en cuanto a las abstracciones del dominio existen paquetes específicos del dominio con abstracciones disponibles para Swarm como librerías para algoritmos genéticos y redes neuronales, aunque ninguno orientado a teorías sociales, pero no se descarta la posibilidad de implementarlos con la herramienta. En relación con las abstracciones organizacionales, Swarm no adopta los conceptos de roles ni organizaciones, principalmente porque su dominio de aplicación no se centra en el trabajo con agentes cognitivos y propuestas centralizadas en la organización., Swarm fue inicialmente desarrollado en el dominio de la *Vida Artificial* (*Artificial Life*, AL o ALife) (Langton 1989; Langton et al. 1992).

Swarm incluye facilidades para el desarrollo como la creación y gestión de interfaces gráficas de usuario o algoritmos para la presentación de datos por medio de histogramas, gráficas, etc.

Cuenta asimismo con librerías para crear modelos con componentes espaciales. Por ejemplo, se puede definir el entorno de un modelo como una rejilla de dos dimensiones, entre muchos otros.

Una dificultad de esta herramienta para su uso es su lenguaje de programación. Objective C resulta complicado para programadores inexpertos. Esta complicación contribuye a la dificultad de uso de la herramienta y a su curva de aprendizaje. Aunque Swarm fue extendido para soportar Java, esta extensión es a través de una interfaz que se ejecuta por encima del kernel de Swarm, por lo que el lenguaje original sigue siendo Objective C y la interfaz se tiene que adaptar a éste perdiendo flexibilidad y facilidad de uso. Por otro lado, aunque cuenta con librerías que proporcionan la infraestructura necesaria para la experimentación y gestión de datos, la mayor dificultad concerniente al diseño de los agentes no es soportada realmente por la herramienta. La abstracción de agente que proporciona es muy simple. Se basa prácticamente en un objeto que puede programar sus acciones en un *planificador* predefinido. No cuenta con ninguna arquitectura interna de agente, la cual tiene que ser definida por el usuario.

De las actividades del proceso de desarrollo de la SSBA, Swarm está dirigido a la *construcción y ejecución de los modelos*. El *análisis*, la *verificación* y *validación* se llevan a cabo con las facilidades de experimentación y gestión de datos que proporciona. La comunicación, *publicación* y *replicación* del modelo es en código de programación solamente. Esta es una herramienta de programación que no está dirigida a soportar la especificación de los modelos en un lenguaje abstracto de alto nivel.

Es fundamental remarcar la importancia de esta herramienta en el dominio del modelado basado en agentes. Swarm ha tenido un impacto duradero en cómo las simulaciones basadas en agentes son escritas hoy en día. Muchos de sus conceptos y patrones de diseño han probado ser fundamentales y prevalecen desde entonces como una referencia en la mayoría de las herramientas de este dominio.

3.3.2. Repast

Repast (*Recursive Porous Agent Simulation Toolkit*), originalmente desarrollado por el grupo *Social Science Research Computing* de la *Universidad de Chicago* (Collier et al. 2003), es un marco de programación de código abierto para el desarrollo de simulación basada en agentes usando el lenguaje Java. Está constituido como una librería de clases para crear, ejecutar, monitorear, desplegar y gestionar datos de las simulaciones. Es decir, permite a los programadores construir entornos de simulación, crear agentes en redes sociales, recoger los datos de las simulaciones de manera automática, y desarrollar interfaces de usuario fácilmente. Sus características y su diseño le deben mucho a Swarm.

Una simulación en Repast consta de *agentes* de cualquier tipo y de un *modelo* que configura y gestiona la ejecución del comportamiento de estos agentes de acuerdo a un *planificador*. La ejecución de la simulación está dividida en pasos de *tiempo* en los cuales los agentes pueden ejecutar alguna acción. La especificación de los agentes y el entorno en Repast se lleva a cabo por un conjunto de variables y métodos según el paradigma de orientación a objetos.

Esta plataforma no cuenta con conceptos de agentes inteligentes ni arquitecturas definidas, por lo que un *agente* suele ser un *objeto* sin ninguna restricción en relación a su arquitectura interna. En cambio, proporciona librerías específicas para implementar funcionalidades como redes neuronales o algoritmos genéticos. Las facilidades de experimentación y análisis que proporciona permiten visualizar y modificar el estado interno de los agentes así como propiedades del modelo dinámicamente en tiempo de ejecución. Las facilidades de desarrollo permiten visualizar una representación animada de la simulación con una gama de entornos espaciales de dos y tres dimensiones. También proporciona otras herramientas más sofisticadas para registrar instantáneas del despliegue de la simulación, herramientas para el soporte de redes sociales, herramientas para el registro y gráficas de los resultados de la simulación.

En realidad Repast es una especificación (Foxwell 1999) de todo este conjunto de características que cuenta con tres implementaciones: Repast para Java (*Repast J*), *Repast .NET*, y Repast para scripts en Python (*Repast Py*). Repast J es la primera implementación de esta especificación y de mayor interés para este trabajo por su compatibilidad y portabilidad entre plataformas. Repast Py es una herramienta de desarrollo rápido de aplicaciones para producir simulaciones en Repast (ya sea Java o .NET) en las cuales el comportamiento de los agentes es descrito usando un subconjunto del lenguaje Python (Lutz y Ascher 1999). Al ser una herramienta de prototipado rápido los servicios para el usuario se presentan de forma visual a través de una aplicación separada. En Repast J el usuario generalmente construye una simulación combinando y extendiendo las clases Java de este marco de programación. En Repast Py el usuario emplea una interfaz gráfica para configurar las propiedades de los componentes de la simulación y mucho de lo que tiene que programarse manualmente en una simulación con Repast J es sustituido por medio de esta interfaz.

En lo que concierne a las facilidades para el modelado, Repast no contiene abstracciones específicas del dominio para el modelado de teorías sociales pues intenta ser una plataforma abierta en la que éstas puedan implementarse extendiendo su librería de clases. Proporciona un gran número de abstracciones de la infraestructura de simulación, por lo que el modelador no tiene que preocuparse de implementar estas cuestiones. Las abstracciones organizacionales no son soportadas.

En cuanto a la facilidad de uso, Repast J es una librería de programación, por lo que el modelado implica tratar con código en su lenguaje de programación. Se podría decir que es fácil de usar si el programador tiene conocimiento de Java. Repast Py es una herramienta más fácil de usar pero la

desventaja es que solamente pueden construirse modelos muy simples, y no quita que sea necesario utilizar un lenguaje de programación como Python.

De las actividades del proceso de desarrollo de la SSBA, Repast se enfoca principalmente a la *construcción y ejecución de los modelos*, aunque también soporta el *análisis* y proporciona facilidades para la experimentación por medio de la cual se pueden *verificar* y *validar* los modelos. La comunicación del modelo es en código Java (o C#), por lo que las actividades como la *publicación* y la *replicación* no son soportadas realmente por esta herramienta. Es importante notar que Repast Py no es una herramienta que soporte la *especificación o el diseño del modelado*, simplemente ayuda a construir más rápidamente un modelo de simulación, pero por debajo sigue estando Repast J cuyo objetivo no es el de soportar esta actividad, de un nivel más abstracto.

3.3.3. Mason

Mason (*Multi-Agent Simulator Of Neigborhoods*) (Luke et al. 2003) es un sistema de simulación de eventos discretos para procesos individuales y una herramienta de visualización escrita en Java, diseñada para ser lo suficientemente flexible para ser utilizada en un amplio rango de simulaciones, pero con un énfasis especial en simulaciones *swarm*[1] de un gran número de agentes. La filosofía en el diseño de este conjunto de herramientas, como Swarm, Repast y Mason es proporcionar una librería de modelos a la cual un programador experimentado pueda fácilmente añadir características para simulaciones simples. Aunque estas tres herramientas se comportan como simuladores de eventos discretos, en realidad son una forma de simuladores por pasos de tiempo discreto, en la cual las unidades de tiempo son *pasos*.

Mason es una herramienta relativamente nueva en el campo del modelado basado en agentes. Fue diseñado conjuntamente por el *Departamento de Ciencias de la Computación* y el *Centro Universitario de la Complejidad Social* de la *Universidad de George Mason*. La librería está orientada a investigadores que requieren llevar a cabo simulaciones, con un gran número de agentes e interacciones, con visualizaciones ocasionales. Esto quiere decir que los modelos no están obligados a contener una interfaz gráfica, éstos pueden o no estar acompañados de un GUI que permita la visualización y manipulación del modelo en dos o tres dimensiones (usando Java3D).

Este marco de simulación no proporciona herramientas de modelado de alto nivel para programadores inexpertos, ni tampoco proporciona características específicas del dominio. Sin embargo,

[1] Swarm Intelligence (SI) es la característica de un sistema en el cual el comportamiento colectivo de agentes (no-sofisticados) que interactúan localmente con su entorno son la causa de que emerjan patrones globales funcionales coherentes. SI permite explorar soluciones a problemas colectivos (o distribuidos) sin control centralizado o la disposición de un modelo global.

deja abierta la posibilidad de añadir módulos que soporten estas características. Mason fue escrito en Java para aprovechar su portabilidad y consta de una arquitectura modular por capas. En la capa del primer nivel se encuentra un conjunto de estructuras de datos como *utilerías* de propósito general. Sobre este nivel se encuentra el nivel del *modelo* con un conjunto de clases que consisten en un *planificador* de eventos discretos, un *generador de números aleatorios* de alta calidad y una variedad de *campos* o *espacios* que contienen objetos y los asocian con una localización en un entorno. Todo este código por sí solo es suficiente para escribir simulaciones básicas que pueden ejecutarse en línea de comandos sin una interfaz gráfica. La tercera capa, la de *visualización,* permite el despliegue de *campos* y un control para el usuario para manipular la simulación. Este conjunto de herramientas de visualización permiten tratar el modelo como una entidad auto-contenida, en cualquier momento puede separarse el modelo de su visualización o puede cambiarse un tipo de visualización por otra. Todos los elementos de esta arquitectura son auto-contenidos y pueden reemplazarse o extenderse fácilmente.

Un *modelo* en Mason está contenido dentro de una sola instancia, definida por el usuario, de una subclase de la clase *modelo* de Mason. Esta instancia contiene por lo tanto un *planificador* y ninguno o varios *campos*. Para Mason un *agente* es una entidad computacional que puede ser planificada para realizar algunas acciones, y que puede manipular el entorno. Un *agente* no está físicamente en el entorno, aunque podría estarlo, en este caso los agentes son referidos como *agentes incorporados*. A diferencia de Swarm o Repast, Mason no programa eventos en el *planificador* para ser enviados a los *agentes*. En su lugar programa a los *agentes* mismos. Los *campos* en Mason relacionan objetos arbitrarios o valores con localizaciones en un espacio. Muchos de los *campos* son interfaces a arreglos de dos o tres dimensiones, el uso de éstos es enteramente opcional.

La facilidad de uso de esta herramienta esta relacionada con el conocimiento de programación en lenguaje Java con el que cuente el modelador. Entre las facilidades del modelado que proporciona se encuentran solamente las abstracciones de la infraestructura de simulación. Como en las demás herramientas, no cuenta con abstracciones del dominio ni organizacionales. Las facilidades de desarrollo y de experimentación y análisis son altamente soportadas por las capas de *utilerías*, *modelo*, y *visualización* de la arquitectura.

Al igual que las otras librerías de programación para el modelado basado en agentes, Mason soporta las mismas actividades del proceso de desarrollo de la SSBA que se han revisado anteriormente. Esta herramienta se concentra principalmente en la *construcción y ejecución de los modelos* de forma bastante flexible. El *análisis*, *verificación* y *validación* de los modelos pueden llevarse a cabo por medio del conjunto de facilidades que proporciona para la experimentación. La comunicación del modelo, la *publicación* y la *replicación* no son soportadas realmente por esta herramienta, estas actividades se valen solamente del código fuente en Java de los modelos.

3.3.4. Netlogo

Netlogo es un entorno de desarrollo rápido de simulaciones basadas en agentes para modelar fenómenos naturales y sociales. Desarrollado inicialmente por (Wilensky 1999) y en continuo desarrollo por el centro *Center for Connected Learning and Computer-Based Modeling* de la *Universidad Northwestern*. Es especialmente adecuado para modelar sistemas complejos desarrollados incrementalmente. Los modeladores pueden crear modelos con cientos o miles de agentes independientes operando en paralelo, lo cual hace posible explorar la conexión entre el comportamiento *micro-nivel* de los individuos y los patrones *macro-nivel* que emergen de la interacción de muchos individuos. Fue diseñado para proporcionar un laboratorio para enseñar los conceptos básicos de la complejidad. Sin embargo, actualmente puede utilizarse para desarrollar aplicaciones más complicadas. La filosofía que aplica Netlogo es que los entornos y lenguajes de simulación basados en agentes deben ser lo suficientemente simples para usuarios principiantes (Tisue y Wilensky 2004) y al mismo tiempo no debe tener límites a lo que usuarios con experiencia puedan desarrollar. Además estos sistemas deben ser capaces de incluir un gran número de simulaciones ejemplo para ayudar a los usuarios a familiarizarse con las herramientas.

NetLogo es un entorno de modelado visual que permite a programadores inexpertos crear modelos más rápidamente que con Java. Netlogo proporciona un entorno gráfico para crear los modelos y personalizarlos por medio de un dialecto del lenguaje Logo (Harvey 1997), que es interpretado y en principio es sencillo y fácil de programar. Hay que notar que la interfaz gráfica no proporciona un lenguaje de especificación de alto nivel, sino entidades gráficas para construir una simulación con los elementos básicos. El entorno gráfico cuenta con un gran vocabulario de primitivas del lenguaje preconstruidas que facilitan la personalización de los agentes y de su entorno. Los *agentes* en NetLogo son llamados "*turtles*" y pueden moverse sobre una rejilla de agentes estacionarios llamados "*patches*". Cada rejilla del entorno se caracterizan como un agente porque pueden llevar a cabo acciones autónomas, por ejemplo acciones de difusión de calor en el entorno. Tanto los "*turtles*" como los "*patches*" son monitoreados por un "*observador*". También proporciona una característica innovadora llamada *HubNet*, la cual permite a un grupo de personas participar interactivamente en las ejecuciones de las simulaciones junto con agentes computacionales (Wilensky y Stroup 1999).

Cuenta con facilidades para el modelado como abstracciones de la infraestructura de simulación que pueden configurarse de forma visual por medio de una interfaz gráfica, pero no cuenta con abstracciones del dominio. Las abstracciones organizacionales tampoco son soportadas, ya que el concepto de *breeds* (una forma de *grupos*) que proporciona está más orientado a contener una lista de agentes que a un concepto organizacional. Las facilidades para la experimentación y análisis son soportadas por las herramientas para la visualización, gestión y control de las simulaciones

que son bastante eficaces. Permiten el despliegue de las simulaciones en 2D y 3D. Es posible monitorear a los agentes para inspeccionar su estado y controlarlos, y proporciona funciones de exportación e importación de datos de las simulaciones así como del estado del modelo en un momento dado. La interfaz con el usuario es bastante amigable así como la presentación de los resultados.

Sin embargo, a pesar de estas facilidades, los modelos de agentes son en general, bastante simples, con modelos cognitivos pobres o inexistentes. Además, no es fácil adecuar la herramienta a modelos de agentes más sofisticados debido a que es un lenguaje propietario que no está disponible para extenderlo. Con NetLogo solamente es posible definir las acciones de los agentes pero no su arquitectura. Igualmente es muy difícil hacer que los agentes interactúen directamente entre sí (generalmente lo hacen a través del entorno). En resumen, esta herramienta es más eficaz y fácil de usar que las librerías de programación siempre y cuando no se requieren agentes muy sofisticados.

Las actividades del proceso de desarrollo de la SSBA que soporta NetLogo son *la construcción y ejecución de los modelos*. La construcción se lleva a cabo con la interfaz gráfica que permite desarrollar un modelo de forma visual. Sin embargo, no se trata de una *especificación del modelo* de alto nivel. El *análisis* y la *validación* son soportadas por las herramientas de experimentación, visualización y presentación de resultados. La *verificación* es difícil de llevar a cabo pues el código que se genera es propietario y no hay manera de hacer pruebas para verificar la correcta transformación del modelo gráfico al código fuente. Sin embargo, esta tarea debe ser realizada por la herramienta misma. La *publicación de resultados* solamente puede llevarse a cabo por medio de gráficas y datos. El modelo puede publicarse como un applet de Java en páginas Web aunque con ciertas restricciones, por ejemplo, no se pueden mostrar gráficos en 3D. Como no se cuenta con el código fuente del modelo la *replicación* en diferentes plataformas es difícil de realizar.

3.3.5. SDML

SDML (*Strictly Declarative Modelling Language*) (Moss et al. 1998) es un lenguaje de programación declarativo con características orientadas a objetos pues está fundado en SmallTalk y basado en una lógica temporal no monotónica llamada KD45 (Konolige 1992). En SDML el conocimiento es representado como reglas y bases de datos y éstos son usados como un mecanismo de razonamiento. Los *agentes* son dotados con *reglas* que determinan su comportamiento y que pueden ser compartidas por otros agentes debido a la orientación a objetos del lenguaje. Como también está basado en un fragmento de otra lógica autoepistémica (FOSGAL) la consistencia formal del modelo está asegurada.

El paradigma de modelado de SDML es declarativo. Primero se describe el problema que se va a estudiar y de aquí se deducen las consecuencias a través de la ejecución del programa. En SDML

el proceso emerge, no se define de antemano como en las herramientas imperativas orientadas a objetos, revisadas anteriormente. Para definir el modelo en SDML primero se implementan un conjunto de *hechos* verdaderos del *entorno* y de las creencias de los *agentes*. El comportamiento de los *agentes* es descrito a través de un conjunto de *implicaciones*. Si un conjunto de sentencias es verdadero (los antecedentes) entonces otro conjunto de sentencias será verdadero (la consecuencia). Cada implicación de este tipo se conoce como *regla*. De esta forma, se implementan los estados del modelo y el proceso emerge, los estados determinan el proceso (Moss et al. 1997). Así, el modelador identifica las entidades y las relaciones que vale la pena implementar, las relaciones entre esas entidades y el mecanismo de inferencia que decide cómo convertir los hechos en la solución del problema.

En SDML el modelo puede ser considerado como una colección de agentes dotados de un conjunto de reglas lógicas asociadas con diferentes niveles de tiempo. La máquina de inferencia ejecuta las reglas verdaderas en cada instante de tiempo de la simulación, es decir, se comporta como un simulador de eventos discretos. Con respecto a los *agentes,* SDML tiene facilidades para implementar agentes cuya representación del conocimiento se corresponda al formalismo de sistemas de producción de reglas. Sin embargo, presenta dificultades para implementar eficientemente algunos métodos algorítmicos y no existe soporte para estructurar las acciones e interacciones de los agentes.

En relación a las facilidades para el modelado, SDML no cuenta con abstracciones del dominio, ni con un lenguaje de especificación de alto nivel. Cuenta con abstracciones de la infraestructura de simulación básicas, como la gestión del paso del tiempo en la simulación. De las abstracciones organizacionales soporta el concepto de *grupo*. En cuanto a las facilidades para el desarrollo, no cuenta con herramientas específicas para gestionar estructuras espaciales de los entornos ni representación gráfica de los agentes. Las facilidades para el *análisis* y experimentación en SDML se proporcionan a través de una ventana por la cual se tiene acceso a todas las reglas que se están ejecutando en la simulación, al igual que a la base de datos de las reglas de los agentes. En cuanto a la gestión de los datos SDML está algo limitado en facilidades gráficas, lo cual complica la *publicación de resultados* y más aún la *replicación,* que está limitada por la especificación del modelo en un lenguaje propietario.

Por lo tanto, las actividades del proceso de desarrollo de la SSBA que soporta son prácticamente la *construcción y ejecución del modelo*, el *análisis*, la *validación* y la *experimentación*. La facilidad de uso puede calificarse como complicada, pues aunque SDML no requiere que los usuarios sean expertos en el lenguaje de programación subyacente, sí requiere que los usuarios aprendan a usar una interfaz compleja como la de SDML que puede ser tan difícil de dominar como un lenguaje de programación.

3.3.6. Sesam

Sesam (*Shell for Simulated Agent Systems*) (Klügl et al. 2004) considera el modelado a un nivel de abstracción más alto que las herramientas anteriores. Proporciona un entorno genérico para el modelado y experimentación con simulaciones basadas en agentes. Su desarrollo se enfoca especialmente en proveer una herramienta fácil de usar para la construcción de modelos complejos, y orientada a investigadores sin una formación en programación (Klügl 2001). Esta herramienta es un entorno de modelado y simulación que combina conceptos de representación de modelos de forma declarativa de alto-nivel y programación visual. Así, consta de un lenguaje de modelado de alto nivel y un entorno para la programación visual basada en este lenguaje. La herramienta se centra en la especificación de los agentes por medio del lenguaje de modelado con el que se define su comportamiento con diagramas de actividad. Básicamente se trata de agentes reactivos.

Lo interesante de esta herramienta es la propuesta que hace para salvaguardar el vacío que existe en cuanto a métodos para pasar de la especificación a la implementación. Para ello propone construir una especificación de alto nivel basada en primitivas gráficas y después transformar esta especificación a un modelo computacional incluyendo información adicional para precisar mejor el modelo de simulación.

Los *agentes* en Sesam contienen un conjunto de variables de estado y un comportamiento que es implementado en forma de diagramas de actividad tipo UML. Las actividades (nodos de un diagrama de actividad) son vistas como scripts que son iniciadas y finalizadas por reglas. Ambos, el lenguaje de especificación, así como el entorno para el modelado y simulación fueron desarrollados basándose en este simple concepto. El marco para la especificación se enfoca en la representación del comportamiento del agente especialmente en relación con otros agentes, recursos y el entorno general. Existe una distinción predefinida entre *agentes* (entidades activas), *recursos* (entidades pasivas) y el *entorno* (el entorno global que rodea las entidades). Las interacciones en los modelos de Sesam consisten de interacciones indirectas, esto quiere decir que éstas ocurren vía cambios coordinados en el entorno (*estigmergia* (Bonabeau et al. 1999)). Por ejemplo, un agente cambia el entorno más o menos deliberadamente, otro agente podría percibir la modificación alrededor de su entorno y adaptar su comportamiento de acuerdo a este cambio. Modelos más complejos de interacción directa entre agentes requieren una especificación detallada de las secuencias de interacción.

Cada actividad del diagrama de un modelo en Sesam tiene asociada acciones que son ejecutadas cuando el agente está en esa actividad. Las reglas son responsables de la terminación y selección de las actividades. Las acciones de las actividades y las condiciones de las reglas pueden componerse de un número extenso de primitivas predefinidas como bloques de construcción. Así, un usuario puede diseñar visualmente el comportamiento de un agente, las estructuras estáticas de un

agente, los elementos del entorno y su configuración. Con esto, es posible modelar visualmente el modelo de simulación basado en agente sin ser necesaria la programación en un lenguaje de programación.

En lo que respecta a las facilidades del modelado, su lenguaje de especificación no cuenta con abstracciones del dominio relacionadas con teorías sociales. No cuenta con abstracciones organizacionales, y cuenta con las abstracciones de la infraestructura de simulación al igual que todas las herramientas de simulación revisadas. Al ser una herramienta más enfocada al diseño del modelo a alto nivel, carece de facilidades para el análisis y experimentación sofisticadas como las facilidades que proporcionan las librerías mencionadas anteriormente. Las facilidades para el desarrollo son de alto nivel, es decir, el diseño y comportamiento de los agentes se realiza visualmente, sin embargo, al final hay que conocer un conjunto amplio de primitivas de la herramienta para poder realizar este diseño, además el manejo de autómatas no es evidente. En general, para quienes no tengan una formación informática, podríamos decir que su facilidad de uso es alta siempre y cuando no se requieren agentes sofisticados. La herramienta está orientada prácticamente a agentes reactivos que interactúan a través del entorno. Para definir agentes con características cognitivas e interacciones directas es necesario extender el conjunto de primitas y agregar módulos en Java.

Las actividades del proceso de desarrollo de la SSBA soportadas son el *diseño*, la *construcción y ejecución del modelo*. El soporte que proporciona para el *análisis* y la *validación* es escaso. El lenguaje de especificación que proporciona es propietario por lo que la comunicación de los modelos no es por medio de algún lenguaje estándar que permita la *replicación*, aunque la *publicación de resultados* puede beneficiarse de los diagramas de especificación para incluir en las descripciones del modelo. La *verificación* la realiza automáticamente la herramienta al generar el código de los modelos de especificación.

3.4. Patrones de Diseño para SSBA

Los lenguajes y herramientas anteriormente revisadas, se basan en un conjunto de patrones de diseño (Gamma et al. 1995) que tienen su origen en la plataforma Swarm y el estilo de programación que ésta promueve. Básicamente, estos patrones proporcionan la perspectiva de una simulación basada en agentes como una colección de *objetos* (agentes) que son conducidos por *acciones* almacenadas en un *planificador* (*scheduler*). Los patrones y los participantes en cada patrón suelen ser los siguientes:

1. *Patrón planificador*. Proporciona una arquitectura para simulaciones de tiempo discreto con estructuras dinámicas y flexibles conocidas como *programas* las cuales son gestionadas por un *planificador*. Los *agentes* que participan en este patrón implementan métodos

para llevar a cabo sus *actividades*. Los agentes pueden crear y planificar *acciones*. Las *acciones* describen invocaciones a métodos en el *modelo*, a otros agentes o a ambos. El *planificador* asocia acciones con el tiempo de simulación. La entidad *modelo* que participa en este patrón proporciona el contenido inicial del planificador, por ejemplo, las acciones que se realizaran durante la simulación. Igualmente, el modelo puede implementar métodos para llevar a cabo actividades (p.ej. en representación del entorno compartido de los agentes). Como parte de estas actividades, existe la posibilidad de crear y planificar nuevas acciones. El modelo contiene además a los agentes del sistema. Bajo este patrón se requiere un esfuerzo considerable para codificar la dinámica del modelo ya que ésta se construye en base a la planificación de todas las actividades de los agentes. Las actividades son modeladas como objetos. Las acciones suelen ser entidades abstractas (por ejemplo en Repast la acción es una estructura de datos que se compone de llamadas a métodos en el modelo o a los objetos que representan a los agentes).

2. *Patrón modelo*. Proporciona una arquitectura para una clase común de simulaciones basadas en agentes que pueden organizarse en rondas, es decir, en pasos de tiempo en los cuales los agentes tienen la oportunidad de actuar. En estos modelos de simulación la dinámica es dada como una función de un modelo de tiempo discreto. Esto es, los estados (de los agentes, del modelo, etc.) cambian en respuesta al paso del tiempo, basándose en el estado previo. Podríamos decir que el *modelo* cuenta con la maquinaria para que funcione la simulación, contiene: a los agentes, definición de métodos para configurar la simulación (es decir, inicializar los parámetros del modelo con valores por omisión) y para construir los componentes de la simulación (p.ej. crear agentes). Los agentes representan a los actores del modelo e implementan normalmente un método *step*, el cual será ejecutado una vez por agente en cada paso de tiempo. El modelo también puede contener su propio método *step* en el cual se pueden implementar órdenes más sofisticados de activación de los agentes. Otro elemento que suele estar presente en este patrón es el relativo a la situación espacial o física de los agentes, la cual impone restricciones en la localización de los agentes. Generalmente, se asume que los agentes más cercanos tienen mayor probabilidad de interactuar o de influenciarse mutuamente que aquellos que están más separados en el entorno espacial. Los modelos de este tipo de entorno habitualmente se construyen usando técnicas de autómatas celulares (esto es, una matriz de celdas cada una de las cuales está en un estado de un conjunto pequeño de estados posibles. El estado de la celda es determinado por reglas simples y depende del estado de las celdas vecinas y su propio estado previo).
3. *Patrón observador*. Este patrón separa el *modelo* y el *observador* del modelo. Así, la lógica de la simulación (el modelo) queda completamente aislada de la forma en que se va a visualizar, esto quiere decir que pueden añadirse diferentes visualizadores gráficos.

Podemos observar la relación de estos elementos en la Figura 11. En esta figura se encuentra el *modelo* el cual generalmente cuenta con un *espacio*. En algunos casos se incluye más de un espacio en una simulación clásica, pero esto se hace con el propósito de facilitar la visualización gráfica de la simulación, estrictamente un agente sólo podría estar en un espacio físico al mismo tiempo, por esto indicamos la multiplicidad de este componente como 1. El modelo también crea muchos *programas*, y sólo un *planificador* que se encarga de gestionar los programas. El modelo activa a los *agentes* que pertenecen al sistema, ambos crean e invocan *acciones*. Por último, los agentes perciben y actualizan el espacio.

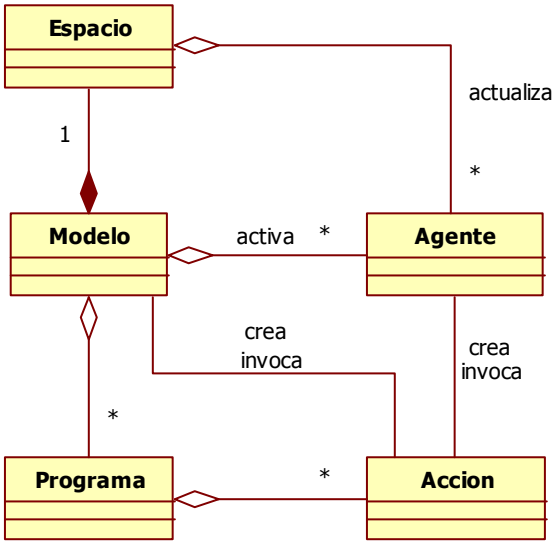

Figura 11. Diagrama de Clases del Patrón de Diseño del Modelado Basado en Agentes Clásico.

Finalmente, una cuestión interesante en el patrón de diseño que proponen las herramientas es el *planificador*. Todas las simulaciones basadas en agentes necesitan una forma de ordenar eventos, aunque con frecuencia esta *planificación* está oculta en el diseño. Una de las motivaciones de utilizar un SMA es su paradigma de control descentralizado que permite que los agentes actúen autónomamente y tomen sus propias decisiones sin intervenciones externas. Si esto último es así, entonces podría parecer contradictorio o inapropiado incluir en un modelo de agentes una entidad *planificadora* como hacen las herramientas revisadas, ya que en éstas, esta entidad funciona como una especie de planificador central.

Sin embargo, ésta no es la función de los *planificadores* de las plataformas de simulación, o al menos no es esta su intención. La definición de un modelo temporal o una planificación está relacionada con el "momento" en el que ocurren las interacciones y no con un planificador "social" que decide quién lleva a cabo que acciones. Es decir, la planificación es una especie de marcador

del tiempo en el cual los agentes tienen la libertad de actuar en un momento dado si así lo deciden, no es un plan de acciones que organice la ejecución del sistema o las tareas que se han de ejecutar (algunas plataformas de simulación consiguen mejor que otras esta perspectiva del *planificador*). Además, el orden de activación de los agentes se procura sistemáticamente de forma aleatoria de un periodo a otro (cada paso) con el propósito de evitar la reproducción de tendencias en los resultados.

Una pregunta que podríamos hacernos cuando hablamos de *planificación* en una simulación cuya aplicación está orientada a observar fenómenos que surgen durante la ejecución de la simulación y que no son "planeados" o programados intencionalmente es: si el comportamiento global de un sistema se dice que emerge de los agentes y sus interacciones, ¿por qué simplemente no se crea una instancia del modelo y se deja a los agentes interactuar y se observa qué es lo que sucede sin la función del planificador? El hacerlo de esta forma ciertamente proporciona un comportamiento emergente, pero los científicos sociales no podrían comprender bajo que circunstancias o que interacciones locales entre los agentes dieron lugar a la estructura global emergida. Además, para descubrir las consecuencias de sus teorías en una SA los modeladores necesitan tener control sobre la simulación para observar qué está sucediendo en cada momento y qué parámetros influencian los resultados.

3.5. Evaluación y Conclusiones

Las SA como método de investigación han mostrado su utilidad para la comprensión de los sistemas complejos y la emergencia. Con éstas es posible hacer evolucionar la abstracción de un sistema en el curso del tiempo con el fin de comprender su funcionamiento a través de la evolución de su comportamiento, observando algunas de sus características dinámicas con el objetivo de evaluar diferentes asunciones sobre el modelo. Sin embargo, el estudio de estos sistemas es complicado, se requieren de metodologías y herramientas para su construcción. El conjunto de metodologías y herramientas de simulación actualmente disponibles imponen un conjunto de restricciones bastante específicas que hacen cada una adecuada para una clase limitada de problemas.

Esto último nos llevó a definir la hipótesis con la que iniciamos esta investigación, qué las prácticas actuales para el estudio de SA deben mejorarse en varios aspectos. Desde la falta de una metodología de desarrollo que involucre actividades como el diseño a través de la especificación de los modelos y su correcta transformación a modelos computacionales de agentes, hasta la necesidad de herramientas más sofisticadas orientadas a la ingeniería de modelos de simulación basados en agentes que soporten esta metodología. Para comprobar esta hipótesis analizamos un conjunto

de metodologías y herramientas del dominio de la simulación social. En el caso de las metodologías tratamos los siguientes aspectos:

- *Su orientación a la construcción de modelos basados en agentes*. Las metodologías actuales adolecen de esta característica en su mayoría. Lo ideal es que la metodología para el estudio de SA utilice el paradigma de agentes en las distintas etapas del desarrollo, no sólo como un lenguaje convenido para la correcta comunicación entre los participantes del desarrollo sino que también se garantice su utilización a nivel computacional.
- *Las actividades que propone para el desarrollo de modelos de SSBA considerando la ingeniería del software*. Las metodologías provenientes de la simulación social están prácticamente orientadas a la investigación y a cómo utilizar los modelos de simulación en un proceso de investigación. Pocas metodologías se centran en los aspectos de ingeniería del software orientada a agentes.
- *La definición de cómo realizar cada actividad y los artefactos producidos*. Las metodologías revisadas carecen de una definición precisa de estas actividades ya que la mayoría son aún propuestas en desarrollo. Cada actividad de la metodología debe ser descrita con precisión para evitar ambigüedades, definiendo cómo y qué debe de producir cada actividad.

El resumen de esta evaluación se recoge en la Tabla 2. En ésta se presentan las metodologías y sus características con respecto a los aspectos ya mencionados, los datos que incluye son:

- *Orientación a agentes*. Si están o no orientadas a agentes y, en caso afirmativo, en qué etapas del proceso de desarrollo se aplican.
- *Actividades del proceso de desarrollo y experimentación*. Qué actividades soporta, empezando por el diseño del modelo, construcción, verificación, análisis, validación, publicación de resultados y replicación. En algunos casos, aunque se menciona la actividad, la definición es muy pobre, por lo que el soporte lo calificamos como *vagamente*.

Metodologías	Gilbert y Troitzsch	Axelrod	Goldspink	Drougol et al.
Orientación a Agentes	No	No	Diseño	Conceptualización Diseño Construcción
Diseño del modelo	vagamente	No	vagamente	Sí
Construcción/Simulación	Sí	Sí	Sí	Sí
Verificación	Sí	Sí	Sí	No
Análisis	Sí	Sí	Sí	No
Validación	Sí	No	Sí	No
Publicación de resultados	Sí	Sí	No	No
Replicación	No	Sí	No	No

Tabla 2. Resumen de las actividades soportadas por el proceso de desarrollo y de experimentación de las metodologías evaluadas y su orientación a agentes.

En relación con su orientación a agentes encontramos que la mayoría de las metodologías revisadas no están orientadas al desarrollo de modelos de simulación con SMA, se centran en prácticas para la construcción de simples aplicaciones de simulación, y en algunas como (Gilbert y Troitzsch 1999) y (Axelrod 1997a) ni siquiera mencionan el concepto de agente en las actividades que proponen. En otras como (Goldspink 2002) el concepto es ambiguo y no se utiliza en todas las etapas del desarrollo, mientras que en (Drogoul et al. 2002) los agentes se consideran en todas las etapas pero falta una especificación del concepto más formal.

En relación a las actividades de la ingeniería de los modelos de SSBA encontramos que las actividades en las que generalmente coinciden todas estas metodologías son las orientadas a la *construcción de los modelos* (implementación), *simulación*, *análisis y validación*, aunque esta última en algunos casos se incluye dentro del análisis y no como una actividad aparte. La actividad como la *publicación de los resultados* es necesaria desde el punto de vista de una metodología de investigación, por lo tanto no en todas las metodologías evaluadas se incluye, (Goldspink 2002) y (Drogoul et al. 2002) no la mencionan. Las actividades que menos se llevan a cabo son el *diseño del modelo*, la *verificación* y la *replicación*. Aunque la *verificación* se menciona en estas metodologías, para que esta actividad fuera una verdadera propuesta tendría que especificarse cómo realizar la transformación del modelo abstracto o formal al modelo computacional, el problema es que tampoco se propone en estas metodologías la creación de un modelo abstracto. La mayoría se centra en la construcción o programación directa del modelo. La única metodología que toma en cuenta el diseño como una actividad necesaria es la de (Drogoul et al. 2002). Por otro lado, solamente la metodología (Axelrod 1997a) propone la *replicación* de los modelos como una actividad de investigación necesaria para que otros puedan confirmar los resultados publicados. La *conceptualización del modelo* se da por sentado en todas las metodologías. Se supone que ya se cuenta con una conceptualización del dominio del modelo real a la hora de iniciar el desarrollo.

En relación con la definición de las actividades y artefactos producidos encontramos que aunque la mayoría de las metodologías define cada actividad, estas definiciones solo dan una idea del proceso de desarrollo. En general, carecen de detalles de cómo realizar cada actividad, cuáles son los artefactos que deben de producirse en cada actividad y cómo están relacionados. Una pequeña excepción a esto es la metodología de (Drogoul et al. 2002), aunque tendría que especificar con más detalle qué artefactos de diseño han de producirse. Asimismo, la metodología (Axelrod 1997a; Axtell et al. 1996) detalla completamente los principios para llevar a cabo la *replicación*, aunque solamente se realiza esta especificación en una actividad de la metodología. Otra carencia de las metodologías es principalmente con qué construir los artefactos producidos, es decir, con qué diseñar o implementar un modelo, qué lenguajes de modelado y qué herramientas son deseables para el soporte de todo el proceso de desarrollo, lo cual no se especifica en ninguna.

Continuando con la comprobación de nuestra hipótesis seguimos con la evaluación de un conjunto de herramientas de desarrollo del dominio de la simulación social. En este caso se trataron los siguientes aspectos:

- *Facilidades para el modelado*. Analizamos dos tipos de herramientas, librerías de programación y entornos de desarrollo rápido. Ambas presentan un soporte limitado, considerando que estas facilidades no solo se refieren a un lenguaje de especificación de alto-nivel sino también a abstracciones del dominio y abstracciones organizacionales. En general, estas abstracciones son limitadas o no se proporcionan. Sin embargo, las abstracciones de la infraestructura de simulación, al ser parte de estas facilidades, representan en general todos los aspectos necesarios para la dinámica de la simulación.
- *Soporte para el modelado de agentes*. Evalúa qué arquitecturas de agentes son propuestas por las herramientas, cómo se representan los agentes y, si no cuentan con ninguna arquitectura verificamos si es posible extender la herramienta para incluirlas. Las herramientas actuales siguen la filosofía de *mientras más simple mejor* (Axelrod 1997a), esto es, que los agentes se mantengan tan simples como sea posible, la complejidad en su diseño sólo se incrementará si es necesario.
- *Facilidades para el desarrollo*. Analiza si se cuenta con componentes que ayuden a desarrollar las simulaciones, es decir funcionalidad que eliminen el trabajo de programación. En general todas las herramientas cuentan con un mínimo de estas funcionalidades.
- *Facilidad de uso*. La facilidad de uso depende del grado o del nivel de conocimiento de programación que se requiera para que el modelador pueda desarrollar una simulación. Aunque las librerías de bajo-nivel pueden considerarse como las más difíciles de usar, entre estas mismas hay varios niveles de complejidad.
- *Facilidades para la experimentación y análisis*. El soporte para la experimentación que proporcionan las herramientas determina el grado de análisis que éstas permiten llevar a cabo. La experimentación la determinamos por las facilidades que proporcionan para observar el comportamiento del sistema, gestionar los datos arrojados por la simulación y por varias ejecuciones del mismo modelo. El análisis además es soportado por diversas utilerías para evaluar estadísticamente los resultados.
- *Comunicación de los modelos*. En principio los formatos utilizados para comunicar los modelos entre diversos participantes de la simulación dependen del tipo de herramienta. Si es de bajo-nivel generalmente es código de programación, si es de alto-nivel generalmente es un formato no-estándar.
- *Actividades del proceso de desarrollo y experimentación que soporta*.

El resumen de esta evaluación se recoge en la Tabla 3 y la Tabla 4. En la primera se presentan las herramientas evaluadas y sus características con respecto a los aspectos de modelado y desarrollo ya mencionados. En la segunda se presentan las actividades del proceso de desarrollo y experimentación de modelos de SSBA soportadas por las herramientas. Los datos que incluyen estas tablas respectivamente son:

- *Lenguaje de especificación*. Si cuenta o no con algún lenguaje para especificar los modelos.
- *Arquitectura de agentes*. Indica que arquitectura se propone, si se conoce el paradigma de implementación, y si se puede definir o extender la herramienta para definir una propia, esto es, *ampliable* o *no-ampliable*.
- *Comunicación entre agentes*. Si la comunicación soportada es solamente a través del entorno se clasifican como *indirecta*, o *directa* si es entre agentes. Si es posible definir mecanismos extendiendo la herramienta para implementar una comunicación directa se clasifican como *ampliable*.
- *Facilidad de uso*. Si se requiere un conocimiento sólido de programación se califica como facilidad de uso *baja*, si se requiere poco conocimiento de programación entonces la facilidad de uso es *alta*.
- *Abstracciones organizacionales, del dominio y de infraestructura*. Para las primera se especifica si cuentan o no con éstas y cuáles de ellas. En las del dominio se especifica si están disponibles o no y si se pueden incluir por medio de *librerías*. Las de infraestructura son siempre soportadas por estas plataformas por lo que este rubro es solo indicativo.
- *Comunicación de modelos*. Si es por medio de código, otro tipo de especificación, ya sea estándar o no. En el caso en el que no se soporta la comunicación es porque el modelo es almacenado en un formato propietario y no es fácil determinar su especificación.
- *Facilidades para el desarrollo y experimentación*. Mientras mayor número de funcionalidades predefinidas proporcione la herramienta más *alta* será su valoración. Puede ser también *media y baja* siguiendo este mismo criterio.
- *Facilidades para experimentación y análisis*. Mientras mayor número de elementos de observación y utilerías de análisis proporcione la herramienta más *alta* será su valoración. Puede ser también *media y baja* siguiendo este mismo criterio.
- *Actividades del proceso de desarrollo y experimentación soportadas*. Simplemente se indica si son soportadas o no. En el caso de la *publicación de resultados,* "*No**" significa que solo se soporta la publicación de los resultados en forma de datos o gráficas estadísticas. En el caso de que también se soportara la publicación de las condiciones bajo las cuales se obtuvieron los resultados, es decir, la especificación del modelo y los parámetros de la simulación, entonces se indicaría con "*Sí*". Cuando ninguno de los dos es soportado o si el so-

porte para la generación de gráficas estadísticas es muy pobre se indica con "*No*". También puede darse el caso que se soporten ambos pero que la especificación sea no estándar, lo cual se indica como "*Sí*$^{no\text{-}estándar}$". Para la actividad de la *replicación* esta se indica como "*No*" soportada mientras no se cuente con algún lenguaje de especificación. En caso contrario, se indica "*Sí*" si es soportada de forma estándar, o en caso que la especificación no sea estándar se indica como "*Sí*$^{no\text{-}estándar}$".

Herramientas	Swarm	Repast	Mason	Netlogo	SDML	Sesam
Lenguaje de especificación	No	No	No	No	No	Sí
Arquitectura de agentes	Plana, objetos, ampliable	Plana, objetos ampliable	Plana, objetos ampliable	Plana, no-ampliable	Lógica no-ampliable	Reactivos, objetos ampliable
Comunicación entre agentes	Indirecta, ampliable	Indirecta ampliable	Indirecta ampliable	Indirecta	Indirecta	Indirecta ampliable
Facilidad de uso	Baja	Media	Baja	Media	Baja	Media
Abstracciones infraestructura	Sí	Sí	Sí	Sí	Sí	Sí
Abstracciones organizacionales	No	No	No	No	Grupos	No
Abstracciones del dominio	No, librerías	No librerías	No librerías	No	No	No
Comunicación de modelos	Código Java	Código Java	Código Java	No	No	Diagramas tipo UML
Facilidades para el desarrollo	Alta	Alta	Alta	Alta	Baja	Alta
Facilidades para experimentación y análisis	Alta	Alta	Alta	Media	Baja	Baja

Tabla 3. Resumen de las herramientas evaluadas y las facilidades de modelado, desarrollo y experimentación con las que cuentan.

Herramientas	Swarm	Repast	Mason	Netlogo	SDML	Sesam
Diseño del modelo	No	No	No	No	No	Si
Construcción/Simulación	Sí	Sí	Sí	Sí	Sí	Sí
Verificación	Sí	Sí	Sí	No	No	No
Análisis	Sí	Sí	Sí	Sí	Sí	No
Validación	Sí	Sí	Sí	Sí	Sí	No
Publicación de resultados	No*	No*	No*	No*	No	Sí$^{no\text{-}estándar}$
Replicación	No	No	No	No	No	Sí$^{no\text{-}estándar}$

Tabla 4. Resumen de las actividades del proceso de desarrollo y experimentación soportadas por las herramientas evaluadas.

En relación a las facilidades para el *modelado, desarrollo* y *experimentación* proporcionadas por las herramientas encontramos que todas siguen los patrones de diseño propuestos por el grupo de desarrollo Swarm (SDG 2006), esto es, una *propuesta simplificada para el desarrollo de mode-*

los de simulación basados en agentes en la cual las simulaciones siempre cuentan con código que conduce la evolución del tiempo, la generación de números aleatorios, herramientas estadísticas y trazador de gráficos. Estos componentes son comunes a todos los modelos de simulación a través de una librería de herramientas que simplifican su diseño, reducen el tiempo de programación y mejoran la fiabilidad del código. Esta adopción puede observarse en el resumen de los resultados de la evaluación de las herramientas en la Tabla 3 en las *abstracciones de infraestructura* y en las *facilidades para el análisis*.

Otro patrón que siguen las herramientas es respecto al esquema del modelo de simulación. Las herramientas proponen un esquema en el que se separa el modelo (el software o el código que simula el sistema) y el observador (un conjunto de rutinas que observan dentro del modelo, recogen estadísticas y las muestran al usuario). Esta propuesta es muy eficiente y tiene mucha similitud con el mundo real, en el cual los fenómenos suceden y los investigadores los observan desde afuera, sin participar en la evolución del sistema. La separación entre el modelo y el observador puede tener una relación con la distinción ontológica sobre la complejidad hecha por (Casti y Karlqvist 1986), es decir, que la complejidad puede no ser una propiedad absoluta del sistema, sino algo que emerge de la interacción entre el observador y lo observado. Esta característica también es común en todas las herramientas, lo cual permite contar con *facilidades para la experimentación* y observación a nivel del modelo y de las entidades que lo componen. En general, las herramientas que están orientadas al modelado de alto-nivel declarativo o de desarrollo rápido como Sesam, SDML y Netlogo tienen pocas facilidades para la *experimentación y el análisis*. Mientras que las de bajo-nivel como Swarm, Repast, y Mason cuentan con una variedad de librerías que proporcionan excelentemente estas facilidades.

Encontramos también que casi todas las plataformas analizadas proporcionan *facilidades de desarrollo*, excepto SDML cuyo entorno de desarrollo está bastante restringido. Cuestiones básicas como gestión de entornos espaciales o estructuras de redes no son soportadas por este. Ninguna de las herramientas proporciona *abstracciones del dominio*, aunque en las plataformas de bajo-nivel éstas podrían incluirse por medio de librerías. Las *abstracciones organizacionales* en principio no se proporcionan en ninguna herramienta por su orientación a sistemas no-organizacionales.

Considerando que las plataformas evaluadas son herramientas para el desarrollo de simulaciones basadas en agentes, incluimos en el análisis la evaluación del tipo de *arquitectura de agentes* que proporcionan, ya sean reactivos, intencionales, o alguna otra arquitectura propuesta. En la evaluación de esta característica encontramos que las herramientas de programación de bajo-nivel como Swarm, Repast, y Mason no cuentan con patrones de agentes predefinidos ni soportan ninguna arquitectura en particular. En éstas, los agentes son simples objetos Java, aunque la arquitectura podría implementarse ampliando la librería de clases. Netlogo tampoco proporciona arquitectura alguna. Además, como el código es cerrado, se desconoce su paradigma de programación de

los agentes, y no es ampliable, por lo que es difícil definir una arquitectura. Sesam y SDML cuentan con una arquitectura más formal aunque los agentes son muy básicos. En el primero podría definirse otra arquitectura, aunque no fácilmente, mientras que en el segundo no. Estos resultados se pueden observar en la Tabla 3.

Otra característica interesante para analizar herramientas es el tipo de *comunicación entre agentes* que soporta. En todas las herramientas solo se soporta una comunicación *indirecta* a través del entorno, probablemente porque los agentes son muy simples, generalmente *agentes tropísticos*, cuyas acciones son determinadas enteramente por su entorno actual, o *agentes histeréticos*, que conservan y son influenciados por su memoria en entornos y resultados previos, así como por su ecología actual (ambos tipos de agentes identificados por (Ferber 1999)). En Swarm, Repast, Mason y Sesam pueden definirse mecanismos de comunicación directa por medio de mensajes, aunque esto debe implementarlo el desarrollador.

Otras características, como *comunicación de los modelos* y *facilidad de uso,* están relacionadas con el tipo de herramienta. Las librerías de programación sólo cuentan con código fuente, Java generalmente, por medio del cual pueden comunicarse los modelos entre los distintos participantes de la simulación o terceros. Netlogo y SDML son de código cerrado o propietario y los modelos sólo pueden comunicarse en el formato de estas herramientas y visualizarse con las mismas. Sesam cae en esta última clasificación, aunque al ser una herramienta de modelado de alto nivel la comunicación es mucho más clara entre personas y un poco más estándar ya que utiliza diagramas tipo UML. La *facilidad de uso* está directamente relacionada con el nivel de programación de la simulación. En general, ninguna herramienta es fácil de usar si consideramos que los usuarios no son expertos programadores. Si se cuenta con nociones de programación, Repast Py, Netlogo y Sesam pueden resultar ideales si no se requieren simulaciones con agentes con procesamiento interno muy sofisticado.

En relación al *lenguaje de especificación*, sólo Sesam considera la necesidad de un lenguaje de especificación de alto-nivel y programación visual basada en este lenguaje. El inconveniente de este lenguaje es que no es estándar (aunque basado en UML) y está limitado a modelos bastante simples, con agentes reactivos.

Finalmente, en relación a las actividades del proceso de desarrollo y experimentación de modelos SSBA soportadas por las herramientas, encontramos que las herramientas de programación de bajo-nivel como Swarm, Repast y Mason soportan mayor número de actividades y pueden extenderse para incluir las que no son soportadas directamente. Los entornos de desarrollo rápido y de alto-nivel como Netlogo, SDML y Sesam cubren menos actividades y son más difíciles de extender debido a que son de código propietario. Esta deducción puede observarse en el resumen de los resultados de la evaluación de las herramientas en la Tabla 4.

De la revisión anterior podemos concluir lo siguiente: que la base metodológica para el estudio de SA es insuficiente pues no cuenta con métodos y prácticas definidas para llevar a cabo cada una de las actividades del proceso de desarrollo. Además, se carece de una orientación al paradigma de agentes pues las definiciones de las actividades se centran en un paradigma clásico de programación. Esta carencia se debe primero que nada a que esta base metodológica se encuentra en desarrollo aún y segundo, y no menos importante, a que no es fácil definir una metodología que integre el conocimiento de métodos, conceptos y prácticas de diferentes dominios como la simulación por ordenador, el paradigma de SMA, las ciencias sociales y la ingeniería del software conjuntamente.

Del análisis de las herramientas podemos concluir que actualmente para construir un sistema de simulación basado en agentes se requiere de mucho conocimiento de programación. En la actualidad las descripciones de modelos encontrados en la literatura de la simulación social son de dos tipos: textuales libres de formato y código fuente. Los primeros tienen la intención de proporcionar una información más precisa de los modelos, el lenguaje que se utiliza varía desde un lenguaje ambiguo a uno más formal adecuado al dominio del problema, aunque antes se requiere una introducción a su terminología. Los segundos son escritos para implementar el modelo y ejecutar la simulación en una plataforma, son menos ambiguos pero mucho más difíciles de comprender por terceros que no conocen la lógica que se ha seguido en la programación. Un tercer tipo de descripción está emergiendo y es en forma de un lenguaje de especificación cuya idea sería acabar con la ambigüedad de las descripciones textuales y la complejidad de las descripciones en código fuente.

Sin embargo, las herramientas actuales en el dominio de la simulación social aún distan de proporcionar un lenguaje de especificación de alto nivel orientado a la definición de modelos de simulación basados en agentes. En realidad existe un vacío en la definición de lo que es un *agente* no solo en el nivel de especificación sino también en los otros niveles requeridos para diseñar una simulación, como el nivel computacional. Esto puede deberse a que a diferencia de las simulaciones tradicionales basadas en ecuaciones matemáticas en las cuales el conocimiento se representa con variables y sus relaciones, el modelado basado en agentes permite, en teoría, una amplia variedad de representaciones más complejas que requieren saber cómo traducirlas a un lenguaje de especificación y luego computacionalmente, y esto último dependerá de la arquitecturas del SMA y del lenguaje usado, lo cual no es trivial.

Este vacío en el concepto de agente genera incompatibilidades entre los niveles y los participantes del desarrollo de un sistema de SSBA y se refleja a nivel computacional, pues actualmente se implementan agentes con arquitecturas poco "comprometidas" con el paradigma de agente para mantener una consistencia de la noción de agente entre todos los niveles. La consecuencia de esto es que no hay una garantía en que lo que se ha diseñado por el experto del dominio corresponda con lo que se ha implementado. Igualmente, para mantener esta misma noción entre los participantes, ocurre lo contrario a un nivel más abstracto, es decir, los modeladores definen conceptos "dé-

biles" de agentes (en lo que concierne al paradigma de SMA) con características consensuadas dada su formación. Generalmente se describe un agente como una entidad que "interacciona", "autónoma" y "proactiva", aunque hemos visto en la evaluación de las herramientas que estas características no se traducen a propiedades computacionales, al menos no son directamente soportadas por las herramientas.

En resumen, es evidente que tanto las metodologías de desarrollo como las herramientas para la SSBA subestiman la mayoría de las dificultades encontradas en la construcción de modelos de simulación social basados en agentes y son insuficientes para dirigir y soportar su construcción, respectivamente. Para que éstas puedan considerarse eventualmente, es necesario adaptarlas para que satisfagan las necesidades planteadas al inicio de este capítulo.

En base a las observaciones anteriores, este trabajo de tesis plantea abordar la problemática anterior proporcionando un *marco metodológico para el estudio de SA* que reúna los siguientes aspectos:

1. *Guía de desarrollo*. Proceso metodológico para el diseño e implementación de modelos de SSBA. El objetivo es utilizar alguna metodología de la ingeniería del software orientada a agentes, especializada en SMA para partir de un modelo más completo de qué es un agente y un SMA, proporcionando recomendaciones y prácticas por medio de una guía de desarrollo para la construcción del sistema.
2. *Lenguaje de modelado de SA (LMSA) de alto nivel*. Un lenguaje de modelado flexible para representar aspectos macro como la organización y aspectos micro como agentes simples a complejos con intencionalidad. Además, debe considerar la representación de aspectos estructurales y dinámicos de la organización y de los agentes. Por ejemplo, estructurales como es el caso de los grupos en la organización, y en el caso de los agentes su estado mental. Los dinámicos pueden ser flujos de trabajo en una organización y en el caso de los agentes la evolución de su estado mental. Este lenguaje debe contar con un nivel de abstracción elevado para la especificación de los aspectos anteriores. El lenguaje será adecuado al dominio de la simulación social. Tanto el lenguaje de especificación como las herramientas requerirán ser adaptadas a las necesidades y conceptos de los modelos de simulación. La idea es proporcionar un marco metodológico fácil de usar, en el que sean mínimas las decisiones de diseño e implementación que tenga que tomar el usuario final.
3. *Utilización del mismo paradigma de SMA en todos los niveles del desarrollo que facilite la validación*. Un modelo computacional de agentes que garantice la correspondencia de las propiedades de agentes en el nivel de modelado más abstracto con el nivel de implementación concreto, esto es, en las plataformas de simulación.
4. *Automatización de la generación de código*. Por medio de mecanismos y herramientas reutilizables que soportan el proceso de desarrollo.

5. *Soporte para la replicación*. Las facilidades que se plantean incluyen la re-implementación del modelo computacional utilizando la misma especificación pero generando código automáticamente para diferentes plataformas de simulación, aunque no se descarta la replicación por medio del re-diseño, esto es, generar una nueva especificación en base a un diseño ya existente y generar el código nuevamente.

Con estos objetivos podemos obtener ciertas ventajas con respecto a las propuestas existentes:

1. Desarrollo de SA con un modelo más formal y completo de SMA.
2. Facilidad de desarrollo para los expertos del dominio (con una guía de *cómo* y *qué* debe de realizarse en cada etapa del ciclo de vida del desarrollo).
3. Facilidad de modelado con conceptos más orientados al dominio social y extensible.
4. Permitir a los expertos del dominio enfocar su esfuerzo en el modelado y evaluación de resultados.
5. Reducción de la necesidad de conocimiento de programación a nivel experto por parte del usuario del dominio.
6. Flexibilidad en la utilización de plataformas de simulación.
7. Potenciar la verificación de errores difíciles de observar, por ejemplo errores de implementación, errores de concepción, o por alguna característica propia de la plataforma de simulación.
8. Comunicación de modelos de simulación más comprensibles por terceros, captando los aspectos sociales de la investigación sin ahondar en detalles de implementación.
9. Facilitar la interpretación de los resultados de la simulación en términos del modelo.

Como posibles metodologías de desarrollo de SMA hemos evaluado algunas provenientes de la ingeniería del software orientada a agentes que pudieran servir como base de la propuesta metodológica que realiza esta tesis. Nuestra propuesta se fundamente en aprovechar tanto las plataformas de simulación existentes como las prácticas de la ingeniería del software orientada a gentes existentes. Como se indicó en el capítulo de la introducción, nuestro trabajo se ha centrado en la metodología INGENIAS (Pavón y Gómez-Sanz 2003). En los siguientes tres capítulos se presenta el marco metodológico cuyas características lo hacen adecuado para los objetivos aquí planteados.

Capítulo 4.

Lenguaje de Modelado de Sociedades Artificiales

Un lenguaje de modelado de SA con un nivel de abstracción más elevado aborda dos aspectos que dificultan la adopción de la SSBA como método de estudio. Uno es el que concierne al conocimiento de programación que requieren las plataformas de simulación para desarrollar modelos de SA. El otro es el que concierne a los modelos de agentes, muy sencillos, que estas plataformas proporcionan. Estos dos aspectos no solo exigen que el experto del dominio cuente con conocimientos informáticos sino también conocimiento de cómo modelar agentes, sobre todo inteligentes, para la simulación social. El lenguaje de modelado que se presenta en este capítulo permite aislar al experto del dominio de cuestiones de programación y centrarse en el modelado de SA. Este lenguaje proporciona conceptos de agentes que le resultarán más naturales para modelar aspectos cognitivos de los individuos.

4.1. Introducción

Teniendo en cuenta, tal como se ha visto en el capítulo 3, la simplicidad extrema del concepto de agente en las herramientas existentes de SSBA, planteamos, en línea con (Conte 2000; Castelfranchi 1998b), y teniendo en cuenta el análisis del área de investigación en el capítulo 2, la necesidad de utilizar un concepto de agente más desarrollado, en línea con las propuestas de la Inteligencia Artificial Distribuida y la ingeniería del software orientada a agentes.

Las sociedades están formadas por individuos que toman decisiones y que actúan. Por tanto, hace falta extender los conceptos de agentes puramente reactivos sobre los que se sustentan estas plataformas a conceptos de agentes inteligentes que faciliten la *especificación* de modelos de estas sociedades. Para ello, se requiere un lenguaje con agentes "cognitivos" que presente las siguientes características:

- Interacción social. Que no se relacionen únicamente de forma directa con su entorno, sino también a través de y mediante de la interacción con los demás individuos.
- Interiorización. El nuevo conocimiento explícito que es asequible en el entorno, se usa para ampliar, comprender, profundizar y redefinir su propio conocimiento tácito.
- Estados mentales. Para ser dotados de representaciones y de intenciones.
- Individuales. Que poseen una representación explícita de sus propios intereses y sus acciones están reguladas por éstos.

Un modelo de sistema social, en el cual los agentes que lo componen exhiban estas características, tiende a captar las propiedades de complejidad que estos sistemas presentan, es decir, los efectos imprevisibles y agregados de una población de agentes en un mundo común:

- Las configuraciones y regularidades no intencionales que se encuentran en la sociedad, es decir, la emergencia de propiedades auto-organizativas que surgen de redes de agentes autónomos conectados entre si (Dupuy 1992; Dumouchel y Dupuy 1983).
- La retroalimentación entre el nivel colectivo y el nivel de los agentes individuales cognitivos.
- La adaptación que los agentes pueden exhibir en su afán de auto-organizarse.
- La dinámica que deriva del comportamiento de los agentes y su interacción.

Para desarrollar este concepto de agente cognitivo se ha elegido el lenguaje de modelado de SMA INGENIAS (Pavón y Gómez-Sanz 2003). Este lenguaje, en línea con (Newell 1982) propone un modelo de agente como una entidad autónoma que tiene un propósito y medios para alcanzarlo. Así, el agente se comporta de acuerdo con el *principio de racionalidad*, el cual establece que si un agente tiene conocimiento de que una de sus acciones lo conducirá a alcanzar uno de sus objetivos, entonces el agente seleccionará esa acción. Con esto, el lenguaje permite muy bien la es-

pecificación de propiedades micro, como el comportamiento intencional de los agentes, y macro como la estructura organizativa y su dinámica, características que están presentes en los sistemas sociales. El lenguaje es apoyado por el INGENIAS Development Kit (IDK) con un editor gráfico que puede extenderse para introducir nuevos conceptos de modelado, como, en este caso, conceptos más cercanos a los utilizados por los expertos del dominio social.

Además, INGENIAS fomenta una ingeniería de desarrollo basada en modelos que facilita la independencia del lenguaje de modelado con respecto a la plataforma de implementación. Con este propósito, el IDK permite la definición de transformaciones entre modelos y generación automática de código para plataformas de implementación.

Esta independencia entre modelo e implementación es especialmente importante, cuando un objetivo de esta tesis es permitir a los expertos del dominio abstraerse de detalles de programación y concentrarse en el modelado y análisis de patrones sociales.

Sin embargo, el lenguaje de modelado de SMA INGENIAS requiere algunas extensiones para su aplicación al estudio de sistemas sociales, que no se habían tenido en cuenta, por no ser inicialmente necesarias para el desarrollo de SMA. Esto considera algunos aspectos que son útiles para modelar patrones sociales.

Es por ello, que se ha tenido que extender este lenguaje, incluyendo nuevos conceptos o adaptando otros ya existentes con el propósito de hacerlo más específico para el dominio social.

En consecuencia, el lenguaje extendido para el modelado de SA (LMSA) facilitará el análisis de patrones sociales en base a la especificación del modelo del sistema. Para poder analizar los patrones sociales del sistema, un módulo de análisis será necesario. Este módulo se ocupará de transformar el modelo en código, que será ejecutado en una plataforma de simulación (el diseño de este módulo es descrito en el capítulo 5). Los resultados de la ejecución podrán entonces presentarse como patrones sociales en términos de los elementos del modelo del sistema social.

En las siguientes secciones de este capítulo se presenta la adaptación del lenguaje de modelado LMSA. La sección 4.2 introduce el lenguaje de modelado de SMA INGENIAS, enfocándose en los conceptos que facilitan el modelado de sistemas sociales. La sección 4.3 presenta las extensiones que ha sido necesario incluir al lenguaje de modelado INGENIAS para el estudio de SA. La sección 4.4 discute las ventajas de esta propuesta.

4.2. Lenguaje de Modelado INGENIAS

INGENIAS es una metodología para el desarrollo de SMA con un énfasis en proporcionar herramientas para apoyar todas las actividades del ciclo de vida del desarrollo, desde el análisis a la implementación. Estas herramientas, que están integradas en el IDK, se basan en un lenguaje de

modelado de SMA. Este lenguaje es especificado con un lenguaje de meta-modelado, concretamente GOPRR (Tolvaner 2000). Si se realizan cambios a la especificación del lenguaje de modelado (es decir, a los meta-modelos) las herramientas pueden regenerarse automáticamente. Esto facilita la evolución de la metodología o su adaptación a dominios concretos, como en este caso para el análisis de sistemas sociales.

El IDK ofrece un editor gráfico para trabajar con los modelos de SMA. Una vez que el modelo se ha especificado por el usuario, el IDK proporciona algunos módulos (o *plugins*) que pueden realizar análisis o transformaciones de los modelos. Éstos son generalmente usados para automatizar la generación de código para plataformas de implementación concretas. Se pueden proporcionar también módulos específicos para el análisis de patrones sociales. Como se explicó en la introducción, estos módulos se encargan de la transformación del modelo en código ejecutable para una plataforma de simulación, la ejecución de la simulación, recolección de datos y su interpretación en términos de los elementos del modelo.

El lenguaje de modelado INGENIAS está estructurado en cinco paquetes, que representan las perspectivas bajo las cuales se puede considerar un SMA (véase la Figura 12): Organización, Agente, Objetivos-Tareas, Interacciones y Entorno.

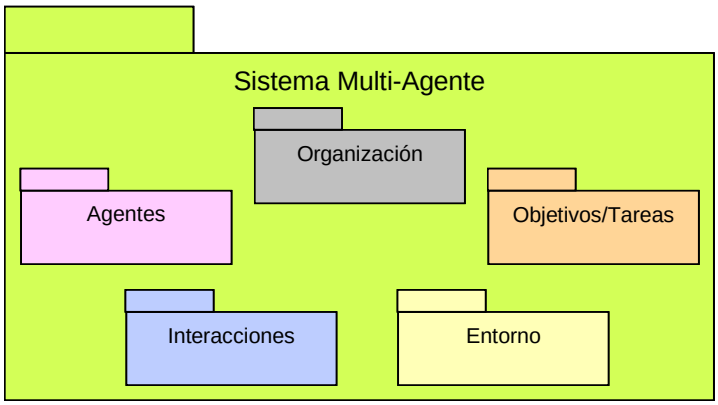

Figura 12. Perspectivas de un SMA según INGENIAS (Pavón et al. 2002)

La organización del SMA determina el marco en el que los agentes, recursos, tareas y objetivos coexisten. Define relaciones estructurales (grupos, jerarquías), normas sociales (limitaciones y formas en el comportamiento de los agentes y sus interacciones), y procesos (en inglés, *workflows*, que determinan cómo colaboran los agentes cuando realizan tareas en la organización).

Los grupos pueden contener agentes, roles, recursos o aplicaciones. Puede haber varias formas de estructurar una organización. Por ejemplo, un SMA puede estructurarse de acuerdo a sus necesidades funcionales, o, al mismo tiempo, los agentes pueden agruparse por distribución geográfica. Un agente, por tanto, puede pertenecer en un momento dado a varios grupos. La asignación de

elementos a grupos obedece a algún propósito organizacional, es decir, porque sus miembros tienen algunas características comunes o porque su agrupación facilita la definición de procesos.

En general, para dar más flexibilidad a la definición de organizaciones se utiliza el concepto de *rol*. Un rol representa funcionalidad o servicios en una estructura organizativa. Los agentes juegan roles en la organización. Varios agentes pueden jugar el mismo rol, cada uno de forma distinta atendiendo a sus capacidades y estrategias.

La funcionalidad de la organización se define por su propósito y tareas. Una organización tiene uno o más objetivos, y depende de sus agentes para llevar a cabo las tareas necesarias para alcanzarlos. Los workflows definen cómo se relacionan estas tareas, y quién es responsable de su ejecución. Los workflows reflejan la dinámica de la organización. Éstos definen asociaciones entre tareas e información general acerca de su ejecución. Para cada tarea, un workflow define cuales son sus resultados, el agente o rol responsable de su ejecución y que recursos se necesitan. Esto es útil para adquirir mayor conocimiento de las relaciones entre agentes a través de tareas, y la asignación y disponibilidad de recursos en una organización.

Ambos aspectos, estructural y dinámico, definen la visión *macro* del SMA. Esta perspectiva facilita la gestión de sistemas complejos ya que permite determinar el contexto y normas de actuación de los agentes, al igual que ocurre cuando se trata de organizaciones humanas.

El comportamiento de los agentes se describe en la perspectiva agente. Este comportamiento viene determinado por el estado mental del agente, un conjunto de objetivos y creencias. También, el agente tiene un procesador de estado mental, que le permite decidir qué tarea realizar, y un gestor de estado mental para crear, modificar o eliminar entidades del estado mental. INGENIAS no explicita cómo se define el procesador de estado mental porque se considera que hay formas muy variadas de realizarlo. Por ejemplo, podría ser un motor de inferencia sobre un conjunto de reglas, un sistema de razonamiento basado en casos, o una red neuronal. Dependerá de las necesidades de la aplicación o el mecanismo más adecuado según el desarrollador.

Los agentes son entidades intencionales, esto quiere decir que actúan porque persiguen unos objetivos. Como además son entidades sociales, colaboran para conseguir satisfacer objetivos de la organización. A la hora de diseñar un SMA se puede empezar identificando objetivos de la organización (del sistema). Estos objetivos pueden refinarse en objetivos más simples hasta llegar a un nivel donde sea posible identificar tareas específicas que puedan conducir a su satisfacción. Otra posibilidad es identificar objetivos individuales para los agentes, que también podrían refinarse de manera similar. En ambos casos, al final habrá una relación entre objetivos y tareas, que es descrita en la perspectiva objetivos-tareas.

Como entidades sociales, los agentes interactúan entre sí. Las interacciones se pueden producir de muchas maneras, siendo las más comunes el intercambio de mensajes, el cual es normalmente asíncrono, la utilización de espacios comunes, donde los agentes pueden actuar (produciendo mo-

dificaciones) y percibir (las modificaciones) como es el caso de pizarras compartidas. Esto se describe en la perspectiva de interacción. En INGENIAS, además de indicar el tipo de mensajes y protocolos en una interacción, otro aspecto fundamental es mostrar la intencionalidad de la interacción: qué objetivos persiguen las partes en una interacción y cómo puede esto contribuir a su satisfacción.

Finalmente, el entorno es lo que los agentes perciben y dónde pueden actuar. Dependiendo de la aplicación, la percepción y actuación tienen significados muy variados. El entorno consiste de un conjunto de recursos, aplicaciones y otros agentes. En muchas situaciones el entorno se puede especificar como un conjunto de interfaces de programación, que serían las clases que lo recubren o que permiten interactuar con él.

4.3. Extensión de LMSA

La simulación de fenómenos sociales implica el modelado de individuos y grupos, y el proceso de interacción social. En INGENIAS, los individuos son modelados como agentes, los grupos y workflows en la perspectiva de organización sirven para modelar la estructura y la dinámica del sistema social. Algunas características de fenómenos sociales, como emociones, presión social, etc., son difíciles de aproximar y pueden modelarse solamente hasta un nivel razonable de abstracción para propósitos específicos.

Actualmente, los patrones sociales bajo consideración en este trabajo de investigación se refieren a la evolución de las creencias y la toma de decisión del individuo en la sociedad (véase por ejemplo el caso de estudio presentado en el capítulo 7 o el estudio sobre la religiosidad en (Pavón et al. 2007a)). Esto requiere la posibilidad de representar la interacción social que da lugar a la emergencia de patrones sociales como cooperación, competición, grupos, organización, etc. Por lo tanto, el nivel de abstracción del lenguaje que usaremos es el de la acción social y la mente del individuo (Castelfranchi 1998a).

El lenguaje para modelar sistemas sociales puede definirse con conceptos ontológicos bajo la perspectiva sociológica de la sociedad en la micro-sociología. Bajo esta perspectiva, un ser humano es conciente de sus acciones y no actúa como reacción a estímulos externos, sino que sus acciones son voluntarias.

De acuerdo a ello, se necesita conceptuar a un *individuo* con *estados mentales* y no simplemente como una entidad reactiva. Estamos de acuerdo con Castelfranchi cuando establece que un individuo debe modelarse como un agente orientado a objetivos cuyas acciones son reguladas internamente por éstos, y sus objetivos, decisiones, y planes se basan en creencias. Los objetivos y creencias son representaciones cognitivas que pueden generarse, manipularse, y ser sometidas a

inferencias y razonamiento internamente. Esta es, básicamente, la propuesta que la perspectiva de agente de INGENIAS proporciona.

No obstante, para contar con un lenguaje de modelado de sistemas sociales se requieren conceptos adicionales al lenguaje de modelado INGENIAS. Algunos aspectos relativos a la perspectiva temporal, localización de los agentes en entornos espaciales, soporte para redes organizacionales y algunas cuestiones de adaptabilidad se añadieron para definir el lenguaje de modelado LMSA. Aunque puede ocurrir que no sea factible definir un lenguaje para el dominio social en su totalidad, sí puede realizarse para uno orientado a aspectos específicos relevantes de ese dominio, como en este caso.

El lenguaje LMSA se basa en el paradigma de agentes software, pero disfrazando la terminología de SMA con los conceptos del dominio social. De esta manera, intentamos hacer más accesible la tecnología a diferentes usuarios, no especializados en la programación, para modelar y estudiar sistemas sociales complejos. Con este lenguaje es posible primeramente abordar uno de los principales problemas de la SSBA que hemos identificado cómo la necesidad de crear especificaciones de alto nivel de modelos de SA. Con estas especificaciones abstractas, la comunicación y comprensión de los modelos será más fácil no sólo para terceros sino también para herramientas que puedan explotar estas especificaciones.

El proceso para la definición de extensiones y generación del lenguaje LMSA que se ha llevado a cabo en este trabajo, incluye las actividades y la participación de los roles que se muestran en la Figura 13.

Los roles que participan en este proceso son el *Rol Experto del Dominio* y el *Rol Ingeniero Software*. La colaboración de ambos roles produce el lenguaje de modelado específico para SA. El Rol experto del Dominio con el conocimiento que tiene del dominio social, define conceptos y las relaciones entre estos conceptos que necesita para modelar procesos sociales. El Rol Ingeniero Software con el conocimiento que tiene de meta-modelado y del IDK, se encarga de adaptar los meta-modelos de INGENIAS en base a las definiciones que ha establecido el Rol Experto del Dominio. Para ello, analiza la definición de extensiones y determina si los conceptos que requiere el experto del dominio se pueden adaptar de los ya existentes en INGENIAS o hay que crear unos nuevos. Este proceso es iterativo pues el Rol Ingeniero Software puede requerir más especificaciones refinadas por parte del Rol Experto. Si todos los conceptos son claros y no se requiere ningún refinamiento, entonces se regenera el editor gráfico del IDK y se produce el lenguaje de modelado LMSA.

A continuación se describen las extensiones que definimos, específicas para el estudio de SA.

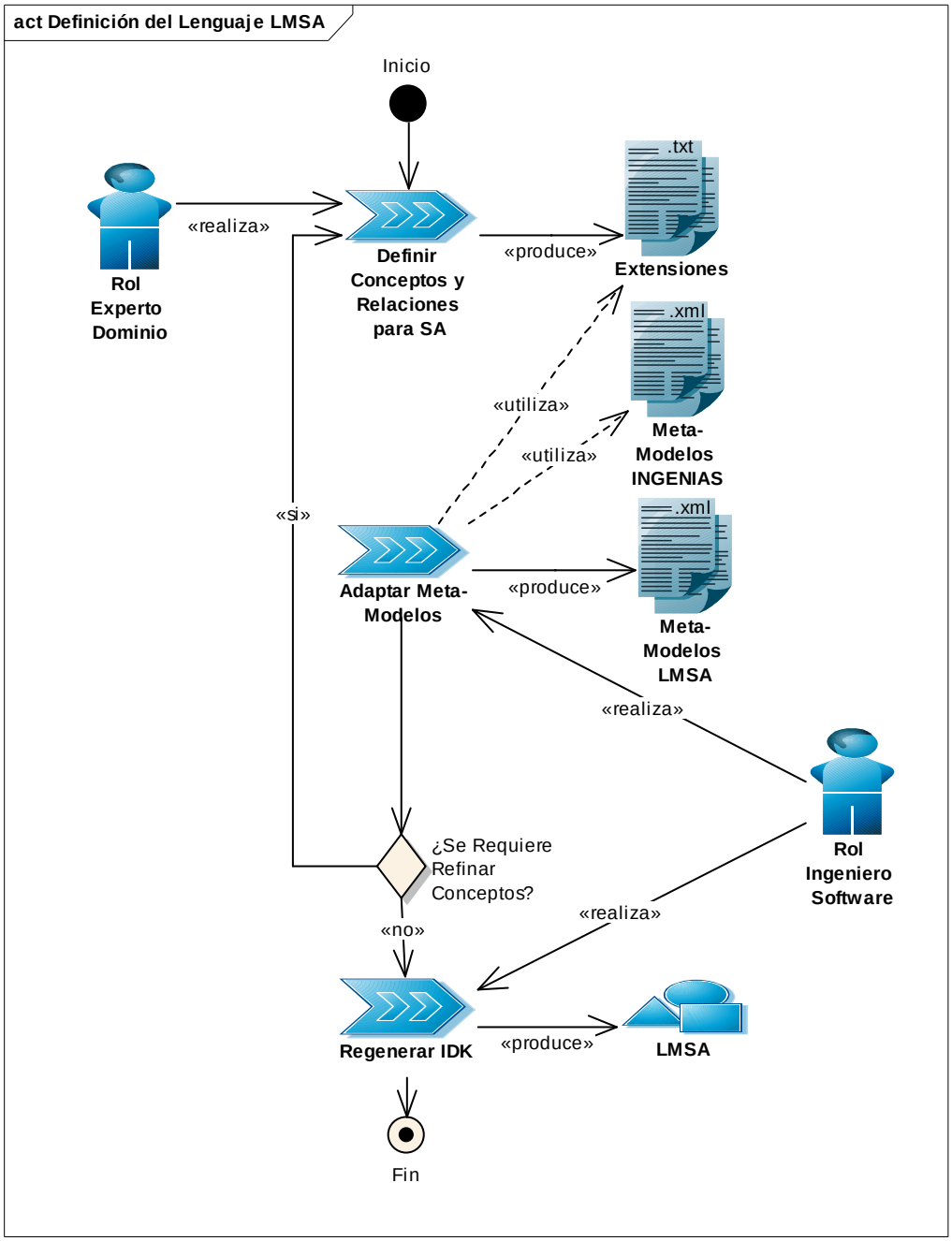

Figura 13. Actividades y Roles en el Proceso de Definición del Lenguaje de Modelado LMSA.

4.3.1.Extensiones

Los aspectos que hemos considerado en la extensión de INGENIAS para la SSBA son los relativos a la *extensión del lenguaje de modelado*. Para ello, ha sido necesario adecuar los meta-

modelos que definen el lenguaje. Las extensiones son de tres tipos: conceptos del dominio social, aspectos relativos a la definición de modelos listos para simular que son difíciles de expresar con el lenguaje de modelado INGENIAS en su estado actual, y perspectivas del modelo que se han de diseñar para realizar una especificación de SA.

a. *Extensiones para la definición de modelos de simulación.* Esencialmente los aspectos a considerar para la extensión del lenguaje, son los relativos a las perspectivas espacial y temporal de las simulaciones. Estos aspectos podrían considerarse como extensiones del meta-modelo de *entorno*.

   1. La perspectiva temporal trata el flujo de tiempo en el modelo durante la ejecución de la simulación. En nuestro caso asumimos que las simulaciones serán dirigidas por tiempo (en vez de por eventos discretos) ya que la mayoría de los entornos de simulación basados en agentes así lo hacen (una razón para este hecho es que un esquema dirigido por eventos requeriría un coordinador de eventos central o una sincronización compleja entre agentes). Por tanto, hace falta modelar pasos de tiempo constantes para simular el ciclo percepción-reacción de los agentes que actúan con el paso del tiempo. Este aspecto es útil cuando se modelan agentes que han de actuar en un momento determinado. La Figura 14 muestra los elementos del lenguaje que se añadieron para realizar la perspectiva temporal de los modelos. Esto se realiza incluyendo una entidad global como una aplicación en el entorno. Las aplicaciones son accesibles a través de un API. Las aplicaciones producen eventos que pueden ser observados. Los agentes definen su percepción identificando los eventos que pueden escuchar. La Figura 14 muestra como un agente percibe el entorno y define su percepción a través de una aplicación en el entorno llamada *Tiempo*. La percepción puede concebirse como la notificación de eventos que llegan a la aplicación (p.ej. pasos de tiempo). Esta representación abstracta de aplicación representa la librería de implementación de las plataformas de simulación que gestionan el tiempo en los modelos de simulación. Así, la perspectiva indica que el agente percibe el tiempo por medio de un método *step* que recibe una notificación de una aplicación *Tiempo* del entorno.

Fig. 14. Extensión LMSA para la Perspectiva Temporal de los Modelos a través de Aplicaciones en el Entorno

2. La perspectiva espacial describe los aspectos relacionados con el posicionamiento de los agentes en un espacio. En general, los entornos de simulación basada en agentes proporcionan espacios de dos y tres dimensiones con configuraciones muy diversas. El modelado de la distribución de los agentes en un espacio se realiza actualmente de forma simple, también a través de una entidad aplicación que representa algunas funciones que proporciona el entorno, como el posicionamiento de los agentes en un espacio en coordenadas aleatorias o en coordenadas x, y específicas. Otra función que proporciona esta aplicación del entorno es que permite a cada agente saber la lista de agentes en el entorno o de sus vecinos. Por ejemplo, para la interacción de los agentes por mensajes es más fácil considerar la lista de los agentes que se encuentran en el entorno que incluir una aplicación directorio para ubicar a los agentes. La Figura 15 muestra los elementos del lenguaje que describen la perspectiva espacial de los modelos. En esta figura puede observarse cómo un agente percibe el entorno como interrogación del estado de una aplicación a través de métodos, es decir, los agentes actúan sobre el entorno invocando estos procedimientos ofrecidos por la aplicación *Espacio*. La representación abstracta de aplicación *Espacio* representa la librería de implementación de las plataformas de simulación que gestionan el espacio en los modelos de simulación. Así, la perspectiva indica que el agente percibe el espacio por medio del método *vecinos* interrogando la aplicación *Espacio* del entorno e invocando el método *getListaAgentes*. Esta aplicación es accesible también a través de un API.

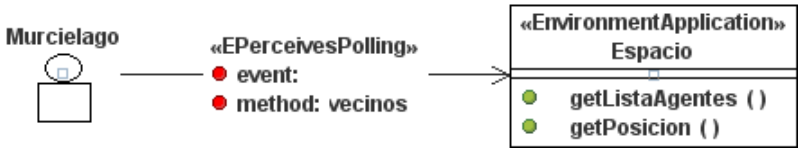

Fig. 15. Extensión LMSA para la Perspectiva Espacial de los Modelos a través de Aplicaciones en el Entorno

b. *Extensiones para conceptos del dominio*. La sintaxis y semántica del lenguaje de modelado INGENIAS en su mayoría cuenta con elementos que pueden adecuarse o corresponderse al dominio social. Solo en algunos casos ha sido necesario incluir nuevos conceptos haciendo modificaciones a los ya existentes.

1. Uno de los conceptos que fue necesario añadir es el de *Red Social*. Una red social es una estructura social dinámica formada por nodos que son generalmente individuos u organizaciones. La red indica las formas en las cuales los actores se conectan dinámicamente a través de relaciones entre ellos, como amistad, lazos familiares, etc. Este concepto está compuesto por lo tanto de *nodos* y *relaciones* entre los nodos, los

cuales se añadieron modificando el concepto de *grupo* de INGENIAS. El que un agente pertenezca a este grupo significa que éste tiene la posibilidad de crear relaciones con otros agentes, y otros agentes pueden crear relaciones hacia este último. Este concepto se utiliza además para modelar parte del mecanismo de adaptabilidad que se ha añadido a los agentes (véase la sección 5.3 y la Figura 23). La Figura 16 muestra la representación abstracta de este concepto en el lenguaje LMSA, la representación a nivel de implementación es una funcionalidad predefinida con las librerías de las plataformas de simulación y las facilidades para gestión de redes sociales. Entre las funcionalidades está el poder formar una red de interacciones y una memoria de éstas para todos los agentes que pertenezcan al grupo *red social* como muestra la Figura 16.

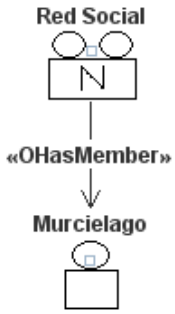

Fig. 16. Extensión LMSA para el Concepto de Red Social

2. También se hizo una adaptación para incluir una entidad llamada *Fuerza Motivacional* que se asocia a los objetivos que persiguen los agentes. Por medio de éste se representa la fuerza (metafóricamente) o *motivación* con la cual son perseguidos dichos objetivos. El fin de este valor es que los agentes puedan cambiar dinámicamente de comportamiento dependiendo de la motivación que tengan (véase la Figura 23). Este concepto se utiliza también para modelar la adaptabilidad de los agentes. La Fig. 17 ilustra cómo este valor de motivación se asocia a un objetivo. Este valor puede cambiar dependiendo de reglas que definen como se refuerza el comportamiento del agente. Según la fuerza *fm* un objetivo puede perseguirse con diferentes niveles de motivación los cuales a su vez pueden estar asociados a diferentes estrategias. En el ejemplo que se muestra en la Figura 17 se muestra que dependiendo del valor *fm* actual al perseguir un objetivo *NG_DarAyuda* un agente puede desempeñar un rol *Martir* que se especializa del rol principal *Altruista*. Así, se pueden especificar diferentes roles o estrategias que puede adoptar un agente dependiendo de su motivación.

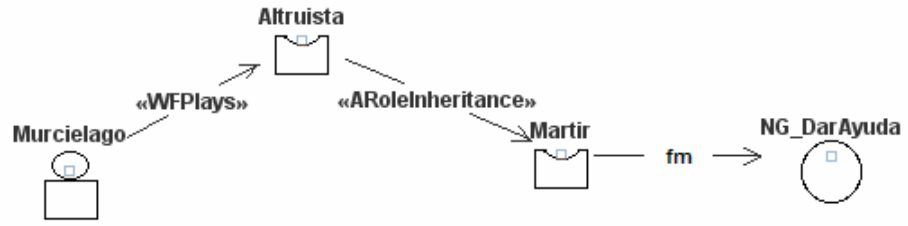

Fig. 17. Extensión LMSA para el Concepto de Fuerza Motivacional

3. El concepto de *Plan* se ha añadido modificando el concepto de workflow de INGENIAS para definir un conjunto de tareas que ejecuta un agente. La diferencia entre estos dos estriba en que, mientras el workflow representa la dinámica que fluye a través de la organización, el plan representa la dinámica del agente mismo. La Figura 18 ilustra la representación abstracta de este concepto en el lenguaje LMSA. Este concepto se utiliza en la perspectiva de objetivos y tareas para definir el conjunto de tareas que componen un plan.

Fig. 18. Extensión LMSA para el Concepto de Plan

c. *Extensiones a las perspectivas del modelo de SMA*. En general se han adaptado una serie de perspectivas que se proponen en INGENIAS para especificar un SMA. Esta adaptación ha consistido en seleccionar los aspectos que son más interesantes para la SSBA, y seleccionar las perspectivas más adecuadas que ayudan a disminuir la complejidad del modelado. Se han añadido dos perspectivas.

   1. La *perspectiva de planes* de un agente, en ésta se describe las tareas que se realizan como parte de un plan del agente. Se utilizan los mismos conceptos de workflow como entradas, salidas y secuencia de las tareas sólo que asociadas al plan del agente. La Figura 19 muestra un extracto de un diagrama con la perspectiva de *planes*. En este caso existe una entidad plan *Día* que se descompone en varias tareas y que conecta otras. Aquí solo se describe la composición de un plan. El plan inicia cuando ciertas precondiciones, expresadas en términos de entidades del estado mental del agente, son satisfechas.

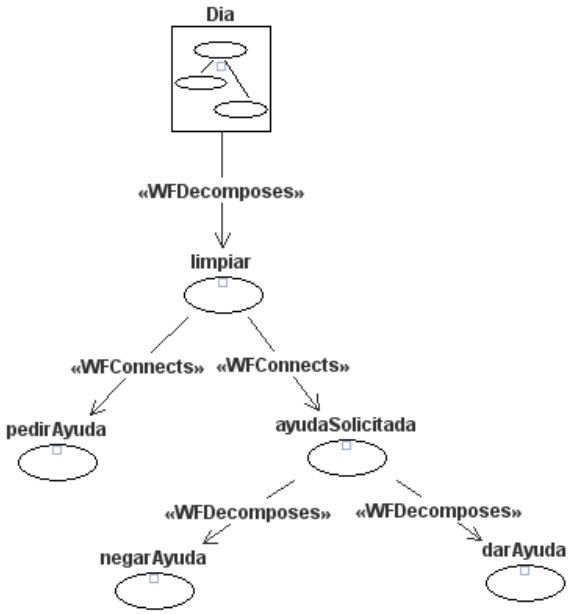

Fig. 19. Extensión LMSA para la Perspectiva Plan del Modelo SMA

2. La *perspectiva de roles y objetivos* para permitir el modelado del mecanismo de adaptabilidad de los agentes. En algunos casos se requieren modelos en los que los agentes cuenten con aptitudes para modificar dinámicamente su comportamiento de acuerdo a cierto *mecanismo de refuerzo*, con esto, un agente puede adaptar sus aptitudes y observar *especialización de roles*. Este comportamiento adaptativo de los agentes depende de la herencia de *roles* y la adición del concepto *fuerza motivacional* presentado anteriormente. Este comportamiento se puede especificar gracias a la ayuda de esta perspectiva y se puede implementar gracias al mecanismo de adaptabilidad (véase la sección 5.3). La Figura 20 muestra el diagrama con los elementos que definen esta perspectiva. Entre estos elementos se encuentra el agente que desempeña determinado rol, la herencia de roles y su asociación a la fuerza motivacional *fm* y los objetivos. La Figura 20 muestra una simple asociación de los roles con la fuerza *fm* por medio de reglas simples. Cuando alguna de estas condiciones se cumple el agente cambia dinámicamente de estrategia.

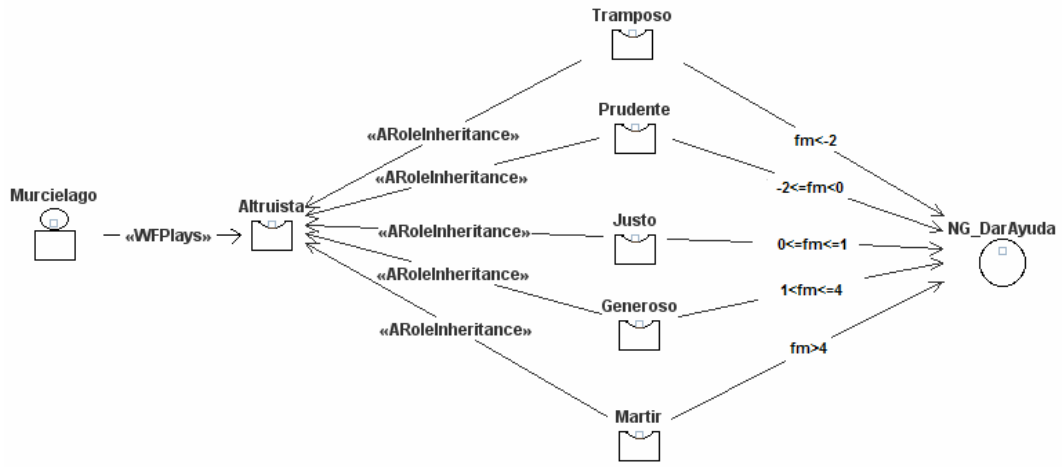

Fig. 20. Extensión LMSA para la Perspectiva Roles y Objetivos del Modelo SMA

Las capacidades de modelado del lenguaje LMSA, incluyendo las extensiones que se han realizado son ilustradas en el capítulo 7, sobre experimentación.

4.3.2. El Editor LMSA

El editor visual que se ilustra en la Figura 14 es el editor del IDK que permite crear y modificar diagramas para trabajar con varias perspectivas de un modelo de SMA. El editor es en realidad la interfaz que permite utilizar el lenguaje LMSA para crear la especificación de una SA en forma de diagramas.

Este editor está asociado a la definición de los meta-modelos de INGENIAS a través de una representación en XML (XML 2006) de los conceptos, de las relaciones, de los diagramas de meta-modelos, así como asociaciones a elementos de representación gráfica de cada concepto. Es por ello, que es posible generar un editor particularizado para el modelado de sistemas sociales realizando las extensiones antedichas a los meta-modelos (en ficheros XML) y a las representaciones gráficas de ciertos conceptos (ficheros de iconos) sin afectar significativamente la funcionalidad del IDK. El editor por tanto podría hacerse aún más específico, personalizando los meta-modelos a las necesidades de un dominio de aplicación más concreto, por ejemplo con conceptos y relaciones de una teoría social específica.

La especificación del modelo de simulación (un proceso incremental e iterativo) una vez realizada, se almacena como una representación XML en un fichero. Este fichero contiene la especificación de todos las perspectivas que componen el modelo de diseño (véase la Figura 41) y constituye una gran parte de la información que será alimentada al mecanismo para la generación de

código (véase la sección 5.4), ya que este modelo representa, a todos los efectos, la aplicación a construir.

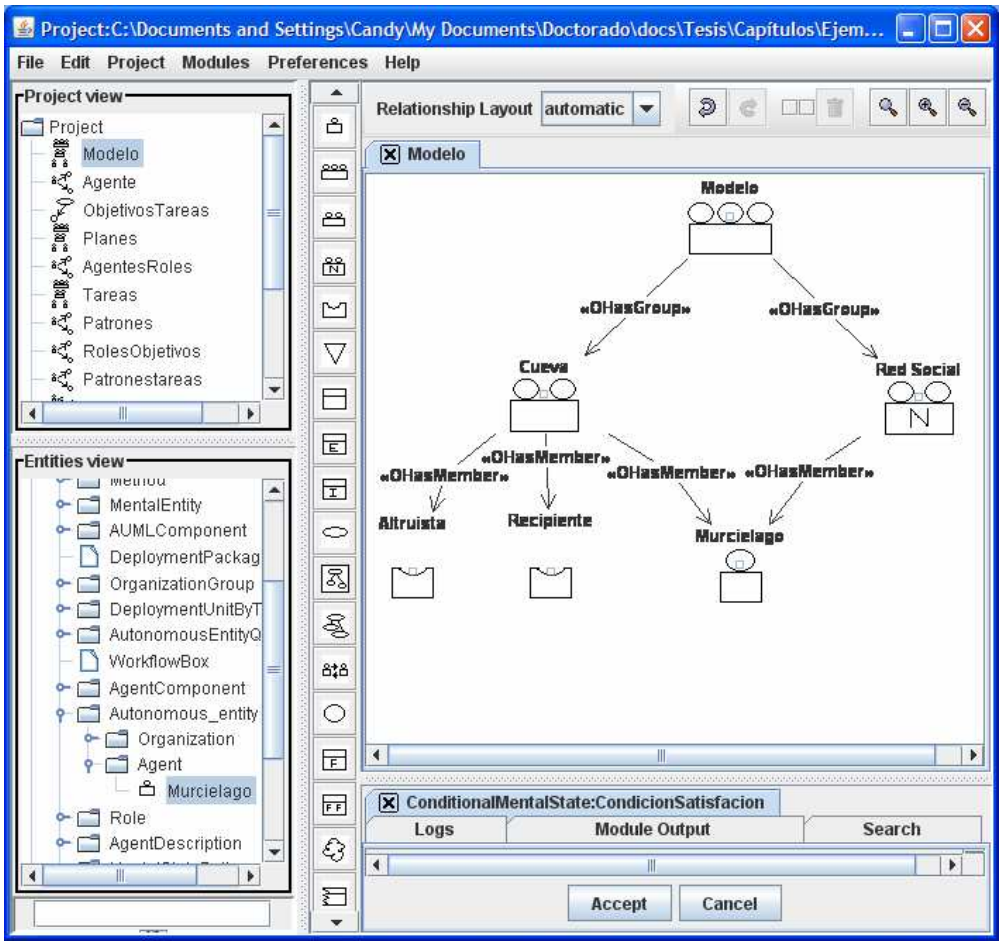

Figura 21. Editor Visual del IDK para la Especificación de Modelos de SA

4.4. Conclusiones

En este capítulo se ha justificado la necesidad de un modelo de agente cognitivo para el modelado de sistemas sociales. Para la especificación de las características intencionales y racionales de estos agentes se ha utilizado el lenguaje de modelado de SMA INGENIAS que ha sido extendido para incluir características particulares de modelado de SA.

La extensión de LMSA incluye abstracciones que permiten especificar un modelo de SSBA visualmente. La especificación con el LMSA es más completa que las descripciones textuales en este dominio lo que conduce a la implementación de dichos modelos más fácilmente.

El lenguaje LMSA es independiente del lenguaje de implementación, lo que permite una representación más cercana al dominio de aplicación, es decir, con conceptos suficientemente comunes al dominio social, y puede extenderse a nuevos conceptos más específicos si fuera necesario. Además, los modelos a este nivel de abstracción son menos sensibles a cambios en las plataformas de simulación subyacentes.

Algunas de las ventajas que proporciona la utilización de este lenguaje son:

1. El usuario pueda trabajar con esquemas visuales, lo cual es más sencillo que trabajar con código para gente que no está iniciada en desarrollo software.
2. El uso de especificaciones visuales permite desacoplar al sociólogo de los aspectos de implementación.
3. Las representaciones basadas en conceptos de agentes cognitivos están más cerca de lo que se espera de una simulación social que involucra el modelado de individuos.
4. Facilita la comunicación de modelos de simulación por medio de modelos más comprensibles por terceros en un lenguaje más abstracto.
5. Facilita la identificación y análisis de procesos y patrones sociales emergentes en términos de los elementos de la especificación del sistema social.

Sin embargo, aunque se trabaje con representaciones visuales, si los sistemas que se quieren simular son complejos (p.ej. número de agentes involucrados, riqueza de la definición del comportamiento individual, etc.), llegado un punto, el nivel de abstracción proporcionado por las especificaciones ya no permite trabajar con comodidad, la especificación empieza a tener un conjunto de diagramas considerable y considera muchos aspectos. Esto significa que aunque trabajemos con especificaciones visuales, se puede llegar a problemas de gestión comparables a los problemas que puede implicar el aprender a usar un lenguaje de programación.

La solución a la problemática anterior es una metodología que ayude a la creación de especificaciones complejas de sistemas sociales (como la que se presenta en el capítulo 6). Esta metodología debe facilitar una transición sencilla de especificación a implementación, ya que los usuarios expertos del dominio social no disponen en muchas ocasiones de personal experimentado para esta tarea. Para esto último, un conjunto de herramientas (véase el capítulo 5) de ayuda al desarrollo serán necesarias para respaldar la metodología.

Capítulo 5.

Herramientas de Desarrollo

Este capítulo está dedicado a la presentación de las herramientas para el estudio y de ayuda al desarrollo que forman parte del marco metodológico que propone este trabajo. Los requisitos de estas herramientas están basados en las necesidades para la SSBA identificadas anteriormente. El proceso de desarrollo que propone el marco metodológico es una guía útil que facilita a los usuarios la construcción de modelos de SA. Sin embargo, una metodología no es suficiente para llevar a cabo la ingeniería de estas aplicaciones, y hacen falta herramientas que faciliten su aplicación. Las herramientas se encargan de impulsar el seguimiento del proceso establecido en la metodología y de aislar a los diferentes usuarios del proceso de desarrollo de la complejidad de ciertas tareas y de proporcionar los medios para llevar a cabo otras con mayor facilidad.

5.1. Introducción

Para que el proceso de desarrollo de SA propuesto en esta tesis pueda adoptarse en el dominio de la simulación social y pueda aplicarse efectivamente es necesario disponer de un conjunto integrado de herramientas que lo soporten. Estas herramientas deberían tener en cuenta varios aspectos. En el caso de la propuesta que hacemos, facilitar las actividades de validación y transformación de los modelos, específicamente a código fuente. Además, según la arquitectura del marco

metodológico, estas herramientas deben cumplir propósitos específicos relativos a los siguientes aspectos:

- *Promover la aplicación del proceso.* Guiar a los usuarios a través de los pasos del ciclo de vida del desarrollo. Ayudar a comprender las tareas y a realizarlas en el orden correcto.
- *Producir artefactos durante el desarrollo*. Ayudan al usuario a construir los artefactos que deben producirse en cada etapa. Pueden ser, por ejemplo, artefactos listos para ser personalizados e instanciados o en otros casos pueden obtenerse por medio de instrucciones para su construcción.
- *Automatizar ciertas tareas*. Tienen como principal propósito facilitar el desarrollo. Una forma de hacerlo es liberar a los usuarios de ciertas tareas complejas o tareas simples pero repetitivas por medio de la automatización de éstas. Por ejemplo, la automatización para la generación de código.

Para la transformación de los modelos conceptuales a modelos de ejecución se ha han elegido dos plataformas de simulación, Repast y Mason (descritas en el capítulo 3). La decisión de utilizar las plataformas de simulación existentes se debe a que consideramos que éstas proporcionan cada una diferentes características que traen perspectivas únicas al campo de la simulación. Por ejemplo, mientras Mason está orientada a simulaciones con un gran número de agentes y distribuidos, Repast proporciona un conjunto de herramientas desarrolladas para la gestión de resultados. Por otro lado, lo que concierne a la generación automática de código, ambas cuentan con características deseables que las hace idóneas para este mecanismo, por ejemplo son librerías Java de código abierto lo cual las dota de gran flexibilidad para ser adaptadas (Tobias y Hofmann 2004).

Además, nuestro propósito es facilitar el uso de las herramientas actuales a los científicos sociales, abstrayéndolos, tanto como sea posible, de cuestiones de programación. Por lo tanto, se propone que los modelos computacionales sean generados para su ejecución en estas herramientas, las cuales han de ser adaptadas para salvaguardar sus inconvenientes.

Para la validación de los modelos la intención de las herramientas es proporcionar un mecanismo que facilite, por un lado la validación de lo que se diseñe sea lo que se implemente, y por otro, que facilite la interpretación de los resultados de la simulación para verificar la validez del modelo con respecto al sistema real. Para la interpretación de resultados se ha aprovechado que la especificación del modelo se realiza con un lenguaje con conceptos abstractos con los cuales es más fácil corresponder los resultados de la simulación para poder observar la aparición de patrones sociales. Para ello, los elementos del código del modelo de simulación deben corresponder adecuadamente con los de la especificación.

En base a los aspectos anteriores, se han desarrollado un conjunto de herramientas que se componen de:

1. Un mecanismo para la generación de código cuyo funcionamiento requiere de plataformas de simulación concretas. El mecanismo se implementa como un módulo del IDK.
2. Un modelo computacional de SMA con agentes cognitivos en línea con el modelo de agentes de INGENIAS. Este modelo computacional se implementa como una librería Java. Esta librería proporciona una abstracción de agentes inteligentes para las plataformas de simulación.
3. Un mecanismo de adaptabilidad para los agentes. La funcionalidad de este mecanismo se implementa como una librería Java. Para los agentes cognitivos no sólo se ha requerido crear un modelo computacional más desarrollado que los que proporcionan las herramientas de simulación, sino también un mecanismo para dotarlos de adaptabilidad. Esto va a permitir simular sistemas sociales con características de auto-organización.

El resto del capítulo se estructura en las siguientes secciones. La sección 5.2 describe el modelo computacional de SMA que se ha diseñado para permitir implementar los conceptos de agentes cognitivos del lenguaje de modelado LMSA, concretamente se presenta el modelo para la plataforma Mason. La sección 5.3 describe el mecanismo de adaptabilidad por refuerzo. La sección 5.4 presenta el mecanismo de transformación de modelos. Se describe el proceso para el desarrollo de un módulo de generación de código, específicamente para la plataforma Repast. Finalmente, la sección 5.5 concluye con una discusión acerca de las herramientas y las ventajas y limitaciones que implica su adopción.

5.2. Modelo Computacional de SMA

La necesidad de abstracciones de SMA, que considere además un modelo de agente inteligente, para las plataformas de simulación se debe a dos aspectos, uno a la transformación de modelos para la generación de código y otro a la interpretación de resultados:

1. El desarrollo MDE que propone el marco metodológico funciona bajo el principio de utilizar el *modelo de diseño* (conceptual) del sistema para derivar su implementación por medio de transformaciones que se realizan sobre éstos. Esta transformación da lugar al código de simulación o *modelo de ejecución*. Para que sea preservada la semántica de los elementos del modelo abstracto de diseño y se asegure que la aplicación resultante sea funcionalmente equivalente a dicho modelo es necesario que el modelo de ejecución de las plataformas cuente de la correspondiente contrapartida a nivel de implementación de estos elementos. Con los modelos de agente de las plataformas de simulación existentes es inviable relacionar los elementos establecidos por el lenguaje de modelado LMSA (utilizado en el diseño) con representaciones software equivalentes. Ello se debe, tal como se vio en el capítulo 3, a

que los modelos de agentes de las plataformas de simulación son muy sencillos y no poseen conceptos de agentes cognitivos como los del LMSA. Así, las plataformas de simulación no pueden cumplir esta función en su estado actual.

2. Para la interpretación de los resultados consideramos que la presentación de resultados directamente en los entornos de simulación es suficiente para el propósito del sociólogo siempre que los elementos del modelo se hayan correspondido adecuadamente con elementos del código del modelo en el simulador (por ejemplo, utilizando los mismos conceptos y nombres para los agentes en el modelo y en el simulador). De esta manera los diagramas de resultados que proporciona el simulador son fácilmente interpretables por el usuario, es decir, el usuario puede llevar a cabo la interpretación de resultados de simulación en términos del modelo.

Por lo tanto, es necesario que las plataformas de simulación puedan disponer de abstracciones de agentes inteligentes, en línea con el modelo de agente definido por el lenguaje LMSA. Además, como el lenguaje LMSA es un lenguaje extendido al dominio social es necesario incluir entre estas abstracciones la correspondiente contrapartida para implementar estas extensiones, como sería el caso de la funcionalidad de los agentes adaptativos.

Para la problemática expuesta anteriormente, cuando los conceptos en los *modelos más abstractos* no corresponden con los conceptos de los *modelos más específicos* (suponiendo que contamos con dos niveles de abstracción por ejemplo), tenemos dos opciones para resolverla: una es transformar o adecuar los conceptos del lenguaje de modelado de alto nivel más abstracto en conceptos existentes en los lenguajes de los modelos más específicos (el segundo nivel menos abstracto), o si no existe del todo una correspondencia clara entre ambos niveles, la siguiente opción es crear un nivel de abstracción más sobre el lenguaje de modelado más específico.

En este trabajo se ha optado por la segunda solución, la de proporcionar un nivel más de abstracción sobre el lenguaje de las plataformas de simulación. Esta opción es la solución más aconsejable ya que el lenguaje de simulación no cuenta con abstracciones del dominio social ni con abstracciones de SMA. Si no lo hacemos así, entonces usaremos agentes y conceptos sociales para modelar SA pero no para implementarlas. Un efecto negativo de esto es que no existe garantía de lo que se ha modelado se ha implementado y que las ventajas que puede proporcionar un modelo de simulación como un SMA no se esté aprovechando.

Por esta razón, incluimos un conjunto de abstracciones sociales y de SMA como un modelo computacional en forma de una librería Java que se adapta a las plataformas de simulación para ampliar los conceptos de su modelo computacional. Estos conceptos se estructuran de tal forma que se amplía el patrón de diseño del modelado basado en agentes clásico (véase sección 3.3.7) pa-

ra añadirle mayor flexibilidad y adaptabilidad. Se añaden abstracciones como agentes racionales, organizaciones, grupos, objetivos, tareas, etc.

En la Figura 22 podemos observar el diagrama de clases del modelo computacional de SMA que se ha añadido a las plataformas de simulación. El modelo se ha diseñado de tal forma que sea reutilizable y que pueda adaptarse a cualquier plataforma de simulación. La Figura 22 muestra el caso específico del modelo para la plataforma Mason (el modelo para Repast es parecido pero adecuado a la infraestructura de simulación de esta plataforma, el detalle de este modelo puede verse en la sección 5.4.2). Para que el modelo de SMA sea reutilizable se ha aislado la infraestructura que es necesaria en todas las simulaciones y que es proporcionada por las plataformas de simulación de las abstracciones de agentes intencionales.

Lo anterior podemos observarlo en la estructura del modelo de la Figura 22. Primero, en la parte de arriba de la figura se presenta el paquete *social.sim.model* que constituye el modelo computacional de SMA y por debajo de este paquete podemos observar los paquetes (*sim.engine*, *sim.field.grid*, *sim.portrayal*) de Mason que proporcionan el soporte para la infraestructura de simulación. Este modelo cuenta con las abstracciones del lenguaje LMSA y las adaptaciones que se le realizaron para el dominio de la simulación social.

El modelo fue diseñado siguiendo algunos patrones de diseño propuestos en (Gamma et al. 1995) para hacerlo reutilizable y flexible. Los patrones de diseño que se han aplicado y que son más adecuados para tratar la problemática que enfrenta el modelo que proponemos son: el patrón de creación *singleton*, el patrón estructural *adapter* y el patrón de comportamiento *template method* (Gamma et al. 1995).

El patrón *singleton* se utilizó para asegurarse que sólo exista una instancia del modelo y que se proporciona un punto de acceso a éste de forma global. En la Figura 22 podemos observar que la clase *Model* es una clase que implementa este patrón y que extiende la clase *sim.engine.SimState* que todas las simulaciones con Mason deben de extender para poder contar con las variables de infraestructura como el *scheduler*. La clase *Model* también cuenta con variables representacionales como la *lista de todos los agentes* de una simulación.

El patrón *object adapter* proporciona una interfaz genérica del entorno por medio de la interfaz *EnvironmentInterface*. El adaptador del entorno es la clase *Environment* el cual adapta los objetos *espacio*, proporcionados por las plataformas de simulación, a la interfaz común *EnvironmentInterface* para gestionar la situación de los agentes en el entorno. En este caso se muestra el objeto *sim.field.grid.ObjectGrid2D* el cual es el *espacio adaptado* a la interfaz del entorno estándar. El último patrón de diseño que utilizamos es el *Template Method* el cual utiliza una clase *abstracta* quel define un *método* que delinea el comportamiento (algoritmo) genérico de los agentes y una clase concreta que implementa las primitivas del algoritmo que define ese comportamiento. Es de-

cir la clase concreta proporciona una implementación de las operaciones abstractas, y puede también redefinir otras operaciones de la clase abstracta.

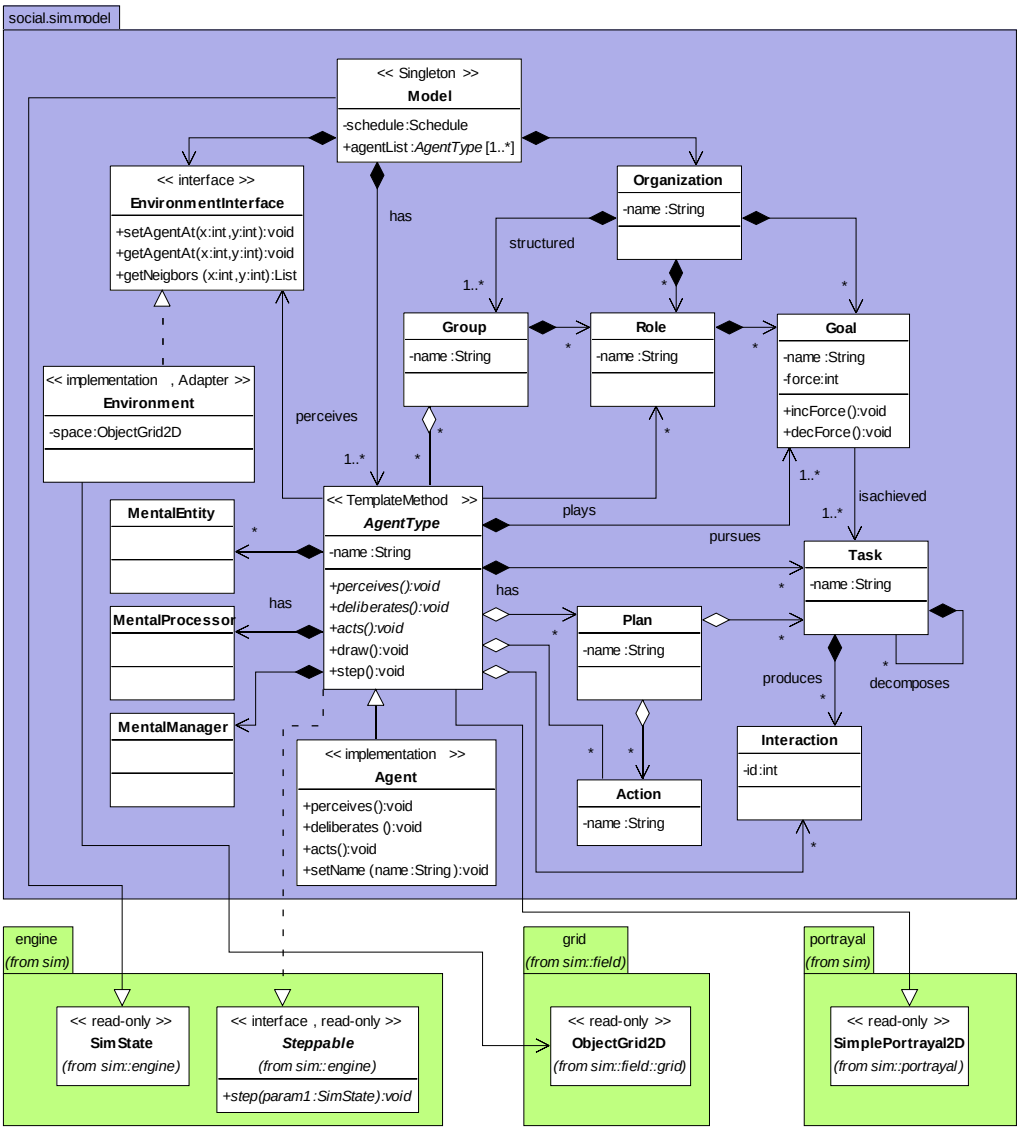

Figura 22. Modelo Computacional de SMA para la Implementación de SA

Por ejemplo, como puede verse en la Figura 22, *AgentType* es la clase abstracta que implementa el método *step* de la interfaz *sim.engine.steppable* de Mason. Este método es un *Template Method* que define la secuencia del comportamiento del agente o mejor dicho, el esqueleto del algoritmo que ha de llevar acabo en cada paso de la simulación, así, la definición de este método esta compuesto de invocaciones a las operaciones abstractas *perceives*, *deliberates*, y *acts*. La subclase *Agent* (la clase concreta de la clase abstracta *AgentType*) implementa estas operaciones según el

mecanismo de control del agente que se utilice. También existen operaciones que proporcionan algún comportamiento por omisión que puede redefinirse si se desea, por ejemplo el método *draw* que implementa como se dibuja un agente en el espacio. Por omisión se proporciona una forma simple de despliegue en dos dimensiones extendiendo la clase *sim.portrayal.SimplePortrayal2D* que es una de las clases de Mason que los agentes pueden extender para ser visualizados gráficamente. Este método *draw* puede redefinirse en la clase concreta *Agent* para proporcionar despliegues más sofisticados.

Por omisión también se proporciona el mecanismo de control de los agentes en forma de un algoritmo que se basa en los objetivos y tareas asociadas a éstos y las evidencias que producen para implementar el mecanismo de procesamiento y gestión del estado mental del agente y su toma de decisiones. Esta es una descripción de la labor del procesador del estado mental del agente tomando como base los *modelos de diseño* que describen la evolución del estado mental del agente. Este mecanismo es sencillo y puede redefinirse con otros mecanismos como algoritmos genéticos, sistemas basados en reglas, etc. Esto se representa en el modelo computacional a través de las clases *MentalEntity*, *MentalProcessor* y *MentalManager*.

En la Figura 22 podemos observar también los conceptos que forman parte de la descripción de un SMA con el lenguaje LMSA. El *Modelo* el cual hereda de la clase *SimState* de Mason es la clase principal del modelo. Un modelo puede contar con organizaciones (*Organization*). La organización estructura el modelo en grupos (*Group*), roles (*Role*) y puede perseguir algún objetivo (*Goal*). El modelo también cuenta con un entorno (*Environment*) y con uno o más agentes (*Agent*). Los agentes pueden organizarse en grupos (*Group*). Los agentes juegan roles, persiguen objetivos, desempeñan tareas (*Task*), las cuales pueden producir interacciones (*Interaction*). La interacción de los agentes se lleva a cabo por medio de *mensajes*. Los planes (*Plan*) son un conjunto de acciones (*Action*) y tareas que llevan a cabo los agentes. Las acciones podemos clasificarlas como habilidades intrínsecas a los agentes, p.ej. enviar mensajes, desplazarse, etc. También pueden clasificarse y definirse acciones propias del dominio de aplicación, por ejemplo para el caso de la SSBA hemos definido una acción que es añadirse como un nodo a una red social.

Finalmente, el mecanismo para la adaptabilidad de los agentes que se ha añadido está representado y soportado por los roles que el agente juega y la asociación de éstos a los objetivos que persigue. Estos objetivos cuentan con un valor o una fuerza con el cual son perseguidos, y dependiendo de este valor los agentes desempeñarán diferentes estrategias o lo que es lo mismo jugarán diferentes roles. También podemos observar que los objetivos cuentan con operaciones para incrementar o decrecer la fuerza motivacional con la que los objetivos son perseguidos. Estas operaciones pueden redefinirse según el algoritmo con el que se implemente el mecanismo de adaptabilidad deseado, en nuestro caso este mecanismo de refuerzo es implementado en base a una *red social* y las relaciones positivas o negativas que afectan el comportamiento del agente.

Con este modelo computacional y los patrones de diseño con los cuales se estructura es posible realizar la correspondencia entre los elementos del lenguaje de modelado LMSA y los elementos del lenguaje de simulación, facilitando así la generación de código y la interpretación de resultados.

5.3. Mecanismo de Adaptabilidad por Refuerzo

Los sistemas sociales suelen presentar una de las principales características de los sistemas complejos, esta es la auto-organización (véase sección 2.3). Un sistema auto-organizativo es un sistema que cambia su organización sin ningún control explicito central interno o externo. El proceso de auto-organización por lo tanto cambia la estructura y comportamiento respectivo del sistema y una organización distinta es auto-producida. A este respecto, el comportamiento social de los individuos es auto-organizado y da lugar a un comportamiento global emergente. Los individuos trabajan con información local a través de interacciones directas o indirectas. De estas interacciones surgen sociedades increíblemente complejas y organizadas. Así, los actores de una sociedad auto-organizativa exhiben una remarcable adaptación a circunstancias cambiantes usando mecanismos para evaluar las opciones a las que se enfrentan.

Para proporcionar más realismo al modelado de actores en un sistema social complejo es necesario por lo tanto dotarlo de estos mecanismos de adaptación. Las propuestas para llevar a cabo este mecanismo varían según el tipo de interacción (desde la interacción indirecta como la *estigmergia* inspirada en sistemas naturales, hasta la interacción directa), arquitectura, etc. En esta tesis hacemos una primera aproximación para proporcionar un *mecanismo de adaptabilidad* para los agentes utilizando una propuesta conocida como *refuerzo*. Con este mecanismo se proporciona la posibilidad de observar auto-organización en los sistemas sociales que se modelan con el lenguaje LMSA del marco metodológico propuesto.

Así, la auto-organización promovida por el mecanismo de adaptabilidad que proponemos esta fundamentada en las aptitudes de los agentes para modificar dinámicamente su comportamiento de acuerdo a algún refuerzo. Éste consiste en los siguientes principios básicos: la recompensa incrementa el comportamiento del agente y los castigos disminuyen el comportamiento del agente (entendiéndose por aumentar o disminuir un comportamiento como reforzándolo) el cual además estará representado por roles. La consecuencia de este refuerzo es que un agente individual puede adaptar sus aptitudes y puede exhibir una especialización de roles a desempeñar. Las aptitudes de comportamiento adaptativo de los agentes dependen por lo tanto de la arquitectura particular de los agentes. En la arquitectura que proponemos, los agentes seleccionan dinámicamente un nuevo comportamiento (o acción) basado en el calculo de un valor que es dependiente del estado actual

del agente y del estado del entorno percibido, así como de la calidad de las decisiones de adaptación previas.

Para definir este valor de refuerzo en los agentes, el mecanismo que planteamos se fundamenta en la dinámica de objetivos de la teoría de la acción social que se plantea en (Castelfranchi 1997; Conte y Castelfranchi 1995; Conte 2000). La dinámica de objetivos es un aspecto esencial de la cognición. En una arquitectura cognitiva un objetivo es una construcción mental altamente dinámica, la cual gracias a las *creencias* puede ser generada, mantenida, promovida, abandonada, suspendida, interrumpida, alcanzada, comprometida, etc. Por lo tanto, la dinámica de los objetivos esta dada por los cambios en las creencias del agente (Castelfranchi y Paglieri 2007) las cuales también influyen directamente en su procesamiento. De acuerdo a lo anterior, necesitamos dotar de dinamismo a los objetivos que persiguen los agentes modificando la arquitectura que proponemos como parte del modelo de agente de INGENIAS. Así, el mecanismo de adaptabilidad por refuerzo de un agente estará compuesto por: 1) la dinámica de los objetivos que persiguen los agentes en base a una creencia que denominamos *motivación*, 2) la adaptabilidad de su comportamiento, la cual es conseguida al especializar los roles que pueden desempeñar los agentes según la motivación que tengan para alcanzar estos objetivos (igualmente la especialización de roles puede implicar que un agente persiga otros objetivos asociados al nuevo rol que desempeña), y 3) la actualización de esa motivación por un *refuerzo* que toma en cuenta sus experiencias pasadas, estado y entorno.

En la Figura 23 se muestran los elementos de este mecanismo de adaptabilidad los cuales proporcionan la funcionalidad anteriormente mencionada.

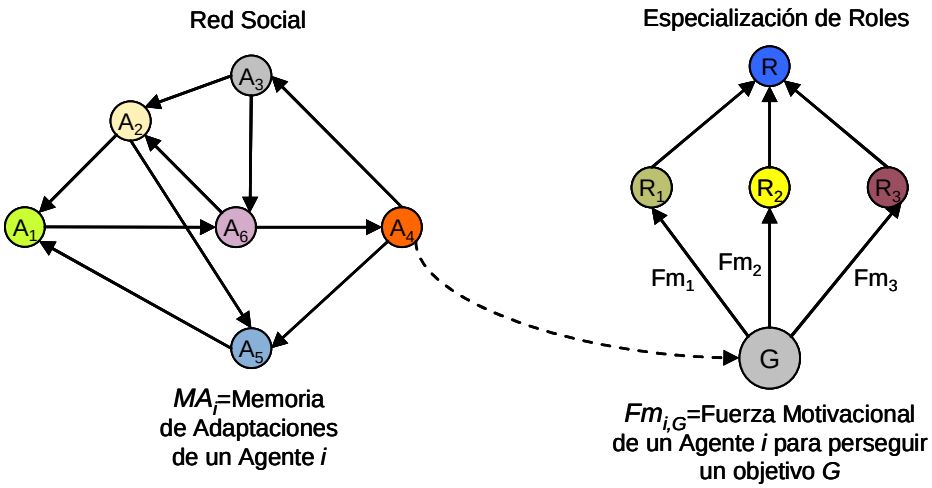

Figura 23. Elementos del Mecanismo de Adaptabilidad por medio de Refuerzo.

Por un lado, los agentes actúan y forman una red de interacción a la que llamaremos *red social*, esta red permitirá que los agentes cuenten con información de lo que llamaremos *memoria de adaptaciones* o *MA*. La lógica de formación de esta red dependerá de la aplicación particular del sistema, como una primera aproximación para este trabajo de investigación esta red representa la experiencia del agente o el impacto positivo o negativo que un agente puede tener sobre otro durante su interacción, conservando esta experiencia en una memoria que podrá reforzar o disminuir su futuro comportamiento. La lógica o *reglas de formación* son definidas por el experto del dominio que lleva a cabo el modelado.

Por otro lado, en la misma Figura 23 también podemos observar que la dinámica asociada al objetivo que persigue un agente esta dada por la variación de un valor de *motivación Fm* (o *fuerza motivacioanal*) para perseguir un objetivo dado. Según este valor, un agente podrá jugar una estrategia o rol diferente dependiendo de la asociación que se haya realizado entre la *motivación* y la estrategia correspondiente como se muestra en la figura. Así, el comportamiento del agente es dinámico dependiendo de su motivación para perseguir un objetivo concreto. *La fuerza de motivación Fm* de un agente *i* para alcanzar un objetivo *G* en un momento dado *t* la definimos por lo tanto como una función de su memoria de adaptaciones previas *MA* que tienen relación con ese mismo objetivo G, su estado interno *S* y el estado del entorno S_E percibido en ese mismo momento de la simulación. Así, para un agente *i* dónde $i \in 1...n$, el valor de su $Fm_{i,G,t}$ esta dado por la siguiente función *f*:

$$Fm_{i,G,t} = f_i\left(MA_{i,G,t}, S_{i,t}, S_{E,t}\right)$$

Esta función puede definirse por medio de *reglas simples* que incrementen o decrecen el comportamiento de los agentes, o puede basarse en *algoritmos* más complicados dependiendo del sistema que se esté simulando. Consecuentemente, un agente individual puede adaptar sus aptitudes y exhibir un comportamiento adaptativo en base a esta función. En el desarrollo de este mecanismo hemos implementado un conjunto de reglas simples que desempeñan esta función y que incrementan o disminuyen el valor de *Fm* como un efecto del impacto de otros agentes en las condiciones propias de otro agente, esto con la ayuda de la red social. Por ejemplo, en el caso de estudio que se presenta en el capítulo 7, los agentes incrementan su motivación *Fm* cada vez que reciben ayuda y la disminuyen cada vez que se les niega ayuda, cada una de estas interacciones tiene un impacto positivo y negativo sobre el agente respectivamente.

Finalmente, para terminar de armar el mecanismo de adaptabilidad por refuerzo, el modelo propuesto se enfoca en definir las relaciones entre diferentes comportamientos, representados por roles, que un agente puede seguir dinámicamente comenzando de su estado actual. Estas relaciones son usadas para seleccionar el nuevo comportamiento del agente cuando se necesite adaptar su comportamiento actual. El comportamiento del agente es descrito en términos de un grafo de com-

portamiento como se muestra en la Figura 24. El grafo de comportamiento incluye los nodos que corresponden a roles los cuales son conectados unos con otros vía enlaces apropiados. Estos enlaces contienen entidades especificando cuando el agente puede cambiar entre los roles respectivos. Con ello, la selección de especialización de roles tiene lugar en tiempo de ejecución basado en estas *entidades condicionales* asociadas con los enlaces del grafo de comportamiento. Las entidades pueden ser condiciones con parámetros representando propiedades de los agentes y de su entorno percibido cuyos valores pueden cambiar dinámicamente durante la ejecución del agente. Aunque de forma más práctica éstas podrían ser una simple asociación de roles con la motivación de los agentes. Sin embargo, estas entidades condicionales (igual que el valor de motivación del agente) pueden realizarse por medio de algoritmos más complejos, por lo tanto, en la Figura 24 se representa la selección de roles no como una simple asociación entre motivación y rol, sino como una función de *Fm*, esto implica que el cambio de rol *Ri* que ha de jugar un agente dependiendo de su motivación puede definirse de forma más adecuada a cada dominio de aplicación definiendo esta función.

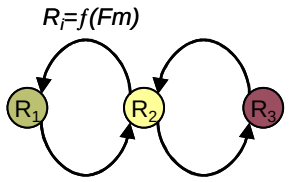

Figura 24. Especialización de Roles por medio de Reglas.

Así, el mecanismo de adaptabilidad por refuerzo que se ha propuesto hace una integración de la técnica de adaptación por refuerzo para la auto-organización la dinámica de objetivos soportada y justificada por las creencias de los agentes (Castelfranchi y Paglieri 2007). El mecanismo por lo tanto implementa el valor de refuerzo de la técnica de adaptación como una creencia que guía el comportamiento del agente. Este mecanismo está orientado a modelar actores sociales con un nivel más elevado de razonamiento, por ejemplo una norma social operará de formas diferentes de acuerdo al nivel de complejidad de la arquitectura del agente. Con agentes simples, dotados esencialmente de un conjunto de reglas de producción y un repertorio de acciones, la salida de una regla de producción será siempre una acción. La intención de este mecanismo es que los agentes sean más inteligentes dotados no solamente de acciones sino también de estados mentales como creencias y objetivos y la dinámica que conlleva esta relación. Por lo tanto, la norma será una construcción más compleja que tomará creencias como condiciones de entrada y proporcionará como salida un conjunto de representaciones mentales como creencias y objetivos. De esta forma, sí de una regla que se ha verificado se infiere que cierto comportamiento debe ser ejecutado, bajo esta perspectiva, la salida principal de esta regla no es la acción o comportamiento que debe ejecu-

tarse, sino una *creencia normativa* que dice que es lo que hay que hacer o lo que se espera, sí se está de acuerdo o no depende de un proceso mental posterior inicializado por la regla, es decir la formación de un *objetivo normativo* basado en una *creencia normativa* (Conte y Castelfranchi 1995). Estas dos representaciones mentales se implementan en el mecanismo de adaptabilidad como la *motivación* y el *objetivo* al cual está asociada tal motivación.

5.4. Mecanismo de Transformación de Modelos

El mecanismo de transformación de modelos proporciona las herramientas necesarias para generar código fuente de modelos de simulación. El mecanismo se implementa como un *módulo* del IDK. Su funcionamiento se basa en tomar como entrada la especificación de un modelo independiente de la plataforma y transformarlo a un modelo específico de la plataforma, en este caso como este último modelo es una especificación en código fuente para una plataforma concreta, representa también un modelo de ejecución.

Los módulos se construyen sobre un marco que proporciona facilidades para recorrer las especificaciones de los modelos, extraer información de éstas, y generar nuevos modelos. En el caso concreto de generación de código se utilizan además plantillas específicas para los modelos de ejecución. El módulo se escribe normalmente en el lenguaje de programación Java siguiendo algunas instrucciones estrictas para permitir su integración con el IDK. Por ejemplo, es imperativo implementar interfaces específicas, algunos procedimientos de extracción de información, la utilización de las plantillas con una determinada estructura, la organización de todos los ficheros en ficheros *jar*, y el despliegue del resultado en una carpeta concreta (véase documentación del API del (IDK 2004)).

La Figura 25 muestra la funcionalidad interna de un módulo generador de código. Al ejecutar el módulo éste utiliza un algoritmo para el recorrido de la especificación del modelo de diseño. Para conocer la estructura y extraer la información necesaria de las especificaciones el módulo utiliza los meta-modelos que definen el lenguaje de modelado, por ejemplo los meta-modelos del lenguaje LMSA. Seguidamente para generar código toma como entrada los elementos que se han aislado al recorrer la especificación y a partir de estos y de plantillas genera el código fuente.

Así, el desarrollo del módulo para la generación de código involucra la generación de plantillas para las plataformas de simulación. Estas plantillas representan aplicaciones prototipo para cada una de las plataformas. Además, en ellas se realiza la correspondencia de los elementos de diseño a elementos de implementación. Esto es, las plantillas contienen por una parte el código fuente en el lenguaje de programación de la plataforma específica, y por otra parte las etiquetas que indican

los lugares donde es necesario añadir la información del modelo. Así pues, las plantillas pueden verse como documentos XML.

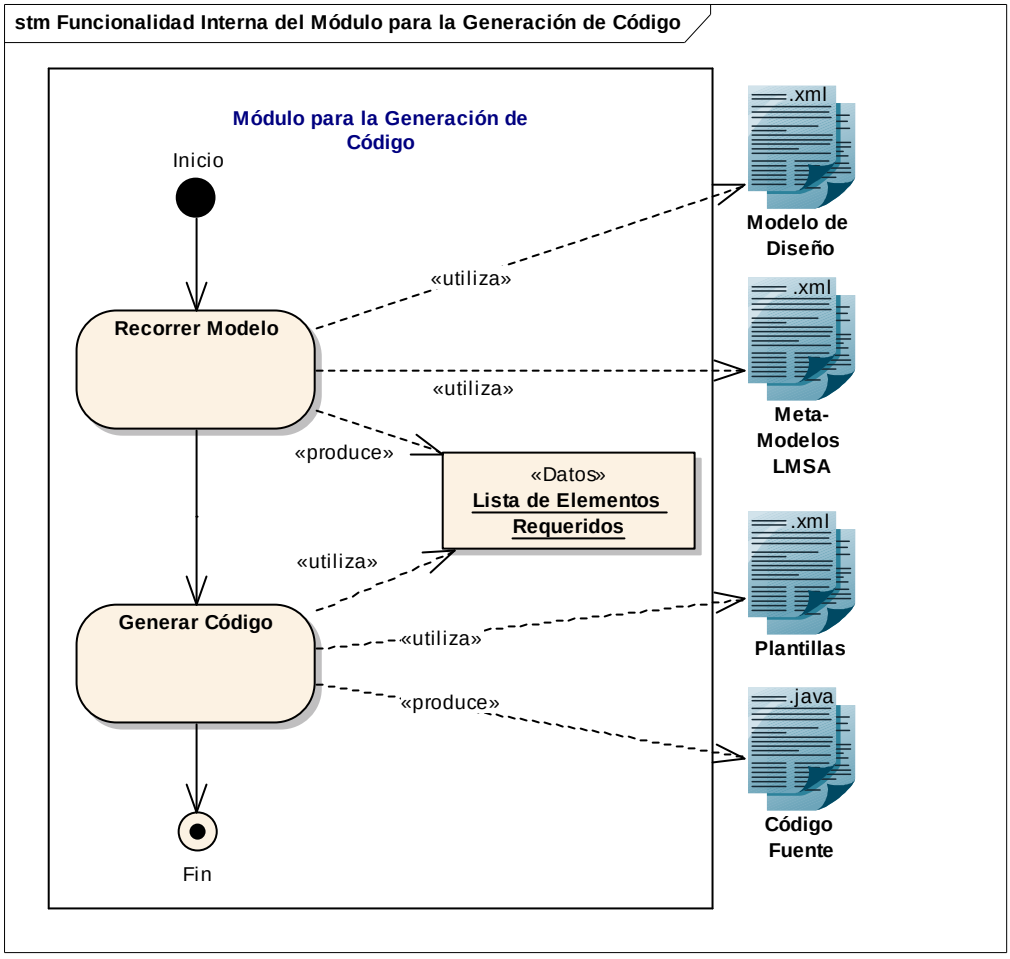

Figura 25. Funcionalidad Interna del Módulo para la Generación de Código.

En el caso concreto de las plataformas de simulación utilizadas en esta tesis, la base de las plantillas es la librería del modelo computacional de SMA presentado en la sección 5.2. Esta librería va sobre las librerías de Repast y Mason, con éstas se crea un prototipo de SSBA cuya funcionalidad se verá completada por el módulo generador de código al sustituir las etiquetas correspondientes.

5.4.1.Proceso de Desarrollo del Módulo

El proceso de desarrollo de un módulo para la generación de código es un proceso normalmente iterativo e incremental de varias etapas. Este proceso va en paralelo con el desarrollo de la aplicación de principal (simulación) cuando la plataforma destino es nueva (es decir, no existe un módu-

lo de transformación para esta plataforma en el IDK). Por lo tanto, mientras se va desarrollando la simulación se va construyendo el módulo.

También, en este proceso de creación del módulo es necesario sincronizar el desarrollo de la especificación con el desarrollo de las transformaciones. Así, más que construir independientemente el modelo de diseño del SMA y las transformaciones de modelos, los dos flujos evolucionan en paralelo con contra-verificaciones frecuentes para asegurar que la transformación actual pueda aumentar para incorporar elementos nuevos de la especificación y para evitar la adición de información que no pudiera implementarse en la entrega final.

De esta forma, los módulos de generación de código son desarrollados de forma iterativa y en paralelo a la especificación del SMA por un lado, y por otro a la programación de la aplicación final sobre la plataforma destino, lo cual es muy importante para definir muy bien las transformaciones. La Figura 26 ilustra estas relaciones, también se muestra en la figura que los puntos de conexión entre los procesos de desarrollo de la aplicación y la definición de la especificación del SMA son las características del modelo que deben ser implementadas. Por *características* se quiere decir partes de la especificación del SMA que tienen un significado por si mismas, por ejemplo, una interacción entre dos agentes. Esta interacción puede especificarse por medio de un diagrama de interacción que contiene los agentes o los roles que participan, la declaración de unidades de interacción (como mensajes) intercambiadas, y las tareas asociadas. Esta interacción puede implementarse sobre la plataforma destino. Para esto, hace falta programar dos agentes que utilicen los servicios de la plataforma para enviar y recibir mensajes, y para ejecutar las tareas. Esta programación facilita el conocimiento de la plataforma y sirve para establecer una primera estructura de código que podrá reutilizarse después.

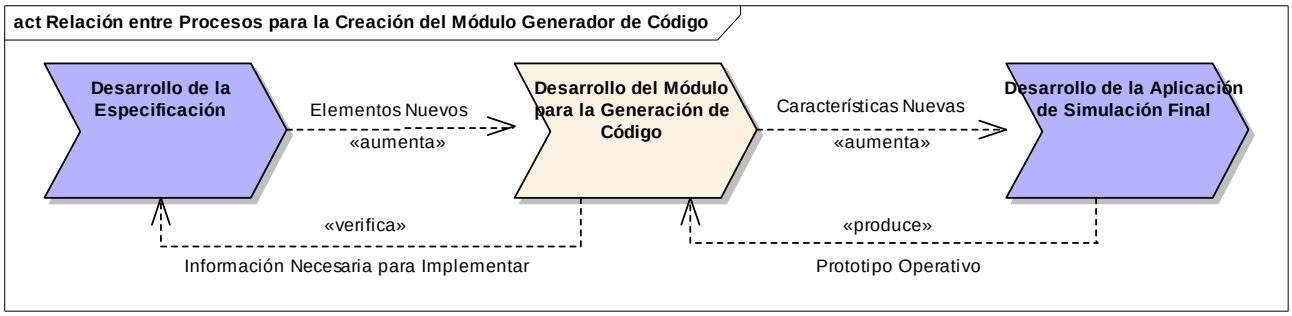

Figura 26. Relación Iterativa entre el Proceso de Desarrollo del Módulo para la Generación de Código y los Procesos de Especificación de SMA y de Desarrollo de la Aplicación.

En efecto, cuando se intente implementar una segunda interacción, es posible generalizar y probar la primera estructura. Esto puede describirse entonces como una plantilla de código y una transformación. Esta transformación identifica los elementos del modelo necesarios para instanciar

la plantilla y rellenarla con los datos apropiados. Es así como empieza el proceso de desarrollo del módulo. A continuación, se toma otra característica del modelo, por ejemplo, la descripción de tareas, y se desarrolla la plantilla y la transformación para implementarla.

De manera ideal, se podría llegar a automatizar de esta forma el desarrollo de aplicaciones y sería simplemente necesario de cambiar la especificación del modelo para modificar la implementación automáticamente con el módulo. Sin embargo, la evolución de las plantillas a través de las diferentes iteraciones es un proceso complejo. Es necesario considerar la redefinición de plantillas, de transformaciones, la integración de componentes software, y la evolución misma de las especificaciones.

Normalmente, la implementación es la parte más difícil de obtener, puesto que esto demanda un buen conocimiento sobre cómo implementar agentes sobre la plataforma destino. Esta dificultad puede aminorarse a través de un proceso iterativo, en el cual se define progresivamente: la arquitectura del código para la plataforma destino, las transformaciones de la especificación, y la generación de las plantillas del código. La Figura 27 muestra este proceso y los roles que participan. En principio son dos, el rol ingeniero software como principal realizador del proceso y el rol experto del dominio que asiste al primero en ciertos pasos relacionados con la especificación del sistema. A continuación se describen los pasos de este proceso.

1. *Crear Prototipo Inicial*. El ingeniero produce un prototipo inicial simple de la aplicación. Al principio, el ingeniero tendrá en cuenta una o varias características de la especificación fáciles a implementar si es posible. Por ejemplo, cómo utilizar las facilidades específicas de la plataforma destino para hacer interactuar dos agentes. Como resultado, el ingeniero ganará el conocimiento sobre la plataforma destino y tendrá un prototipo de una aplicación sobre la misma. Este prototipo implementa ya una parte de la especificación con un conjunto de características elegidas.
2. *Marcar código del prototipo*. Al observar al mismo tiempo el prototipo y la especificación, es posible identificar las partes del prototipo que corresponden a las partes de la especificación. Como resultado, el ingeniero puede definir posibles correspondencias entre la especificación y el código del prototipo. Estas se reflejarán en el código del prototipo marcadas con etiquetas XML. Las plantillas son los pedazos de código fuente marcados.
3. *Generación y modificación del módulo*. Un módulo debe atravesar la especificación para obtener la información necesaria para instanciar y llenar las plantillas del prototipo. Para recorrer las especificaciones se utiliza un algoritmo específicamente creado para la plataforma destino y para las características que se quieren implementar.
4. *Despliegue del módulo*. Las clases y las plantillas del módulo resultantes deben colocarse juntos en un fichero *jar*. Este fichero *jar* se coloca en una carpeta específica donde el IDK puede cargarlo dinámicamente.

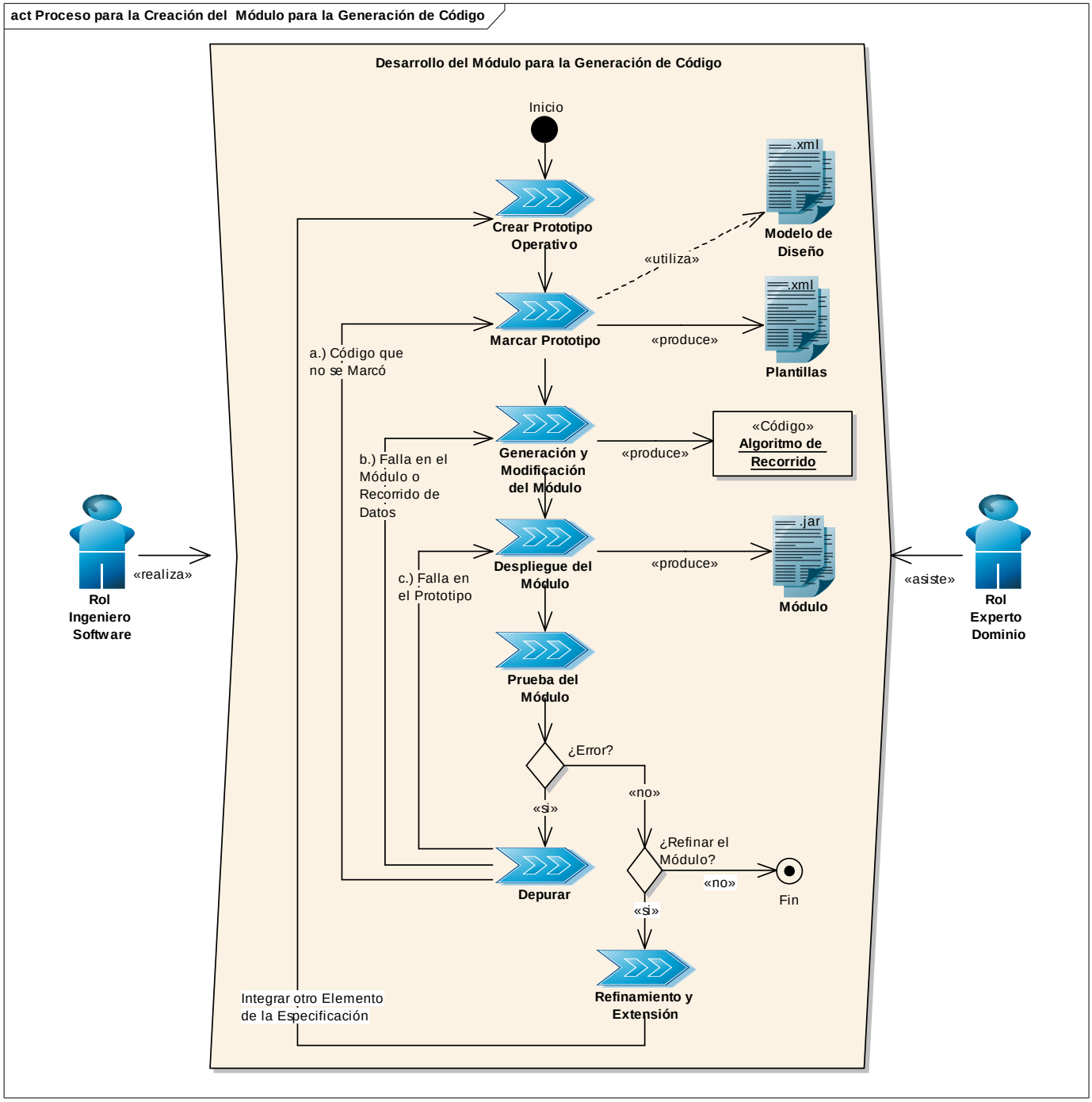

Figura 27. Proceso de Creación del Módulo para la Generación de Código.

5. *Prueba del módulo*. La prueba puede realizarse desde el editor del IDK. Una vez cargado en el IDK, el módulo puede ejecutarse sobre la especificación que actualmente está cargada en el editor. El ingeniero debe examinar si se recorre el diagrama correctamente y si se llenaron

bien las plantillas. Como las plantillas requieren información específica puede suceder que esta información no esté presente o que no esté formulada de la manera adecuada. Puede suceder también que la especificación no sea correcta o esté incompleta. Así pues, un módulo puede ser útil también para comprobar que una especificación esté completa según algunos criterios. Tres tipos de problemas pueden suceder: con el código generado por el módulo, con el recorrido de la especificación, y con la propia especificación.

6. *Depurar*. Si algo no va bien es necesario depurar el prototipo e ir a:
    a. Paso 2. Si hay un nuevo código que no se marco antes.
    b. Paso 3. Si la falla estaba en el módulo y el recorrido de los datos.
    c. Paso 4. Si la falla estaba en el prototipo y puede solucionarse sin marcar el código de nuevo.
7. *Refinamiento y extensión*. Una vez que el módulo se ha terminado, éste puede transformar los diagramas de las especificaciones en código fuente o comprobar la satisfacción de algunas propiedades. El módulo puede haber considerado solamente un conjunto reducido de diagramas. El próximo paso es tomar el código generado por el módulo y extenderlo para satisfacer otras partes de la especificación. Ello significa regresar al primer paso.

Finalmente, el ingeniero software puede producir *scripts* adecuados a cada plataforma de simulación para compilar el código fuente que se ha producido, para desplegar el modelo de simulación en el simulador y para su ejecución si fuera necesario. Estos *scripts* facilitarán estas dos tareas al experto del dominio.

5.4.2.Módulo Repast

Como ejemplo del proceso para el desarrollo de un módulo se presentan a continuación los componentes del módulo que se ha desarrollado para la plataforma de simulación Repast. De un prototipo de una simulación con Repast, podemos aislar una serie de plantillas comunes a la mayoría de las simulaciones. Un ejemplo simplificado de una de estas plantillas puede observarse en la Figura 28. En general son tres los tipos de plantillas:

1. *Plantilla Agente*. Esta platilla implementa un prototipo de un agente SSBA el cual extiende la clase *AgentType* para contar con las facilidades que ésta implementa, por ejemplo la implementación de la interfaz *Drawable* de Repast que permite el despliegue gráfico de los agentes. Esta plantilla se instancia según el número de agentes de la simulación.
2. *Plantilla Modelo*. Esta plantilla implementa el prototipo del modelo que extiende la clase *SimModeloImpl* de Repast y configura los componentes de la simulación, por ejemplo la lista de agentes, el *planificador*, etc. Sólo existe una instancia de esta plantilla.

3. *Plantilla Entorno*. Esta plantilla contiene el prototipo del entorno con un *espacio* por omisión y la posibilidad de deshabilitarlo en el caso que se requiera simular agentes sin una ubicación o referencia física. El prototipo de esta plantilla se basa fundamentalmente en la interfaz *EnvironmentInterface* y la clase *Environment* así como de los objetos que representan *espacios* en Repast. Normalmente sólo existe una instancia de esta plantilla, aunque en simulaciones complicadas puede ser necesario incluir varios espacios en un mismo entorno.

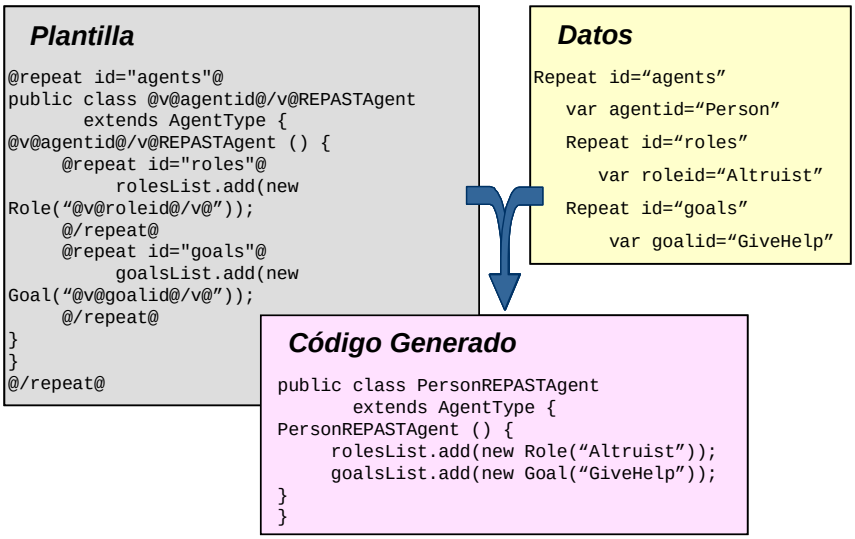

Figura 28. Ejemplo de Generación de Código para un Agente en Repast.

Por un lado el ingeniero software define una plantilla agente de Repast (paso 1 y 2) y extrae los datos requeridos de la especificación del SMA (paso 3) para generar el código. Estos datos pueden incluir por ejemplo el nombre del agente, los objetivos y los roles. Como resultado se obtiene el código fuente Java para el agente Repast SSBA instanciado de la plantilla. Posteriormente todos estos elementos son configurados dentro del módulo y desplegados en el IDK (paso 4) desde donde se realizan las pruebas sobre la especificación SMA (paso 5). La prueba del módulo con la especificación debe producir finalmente el código esperado como en el caso de la Figura 28.

Los prototipos que dan lugar a estas plantillas se desarrollan con el API del modelo computacional SMA y el API de la plataforma Repast. Las siguientes figuras ilustran las entidades de ambas librerías que se utilizan para desarrollar los prototipos que implementan el SMA. Un prototipo de agente y de modelo de simulación en Repast se desarrolla normalmente con los componentes del API que se muestran en la Figura 29.

En la Figura 29 se pueden observar las entidades más comunes con los cuales se desarrolla el modelo de simulación y los agentes dentro de ese modelo con Repast. Todos los modelos de simulación con Repast deben de implementar la interfaz *SimModel*. Repast proporciona una clase abstracta *SimModelImpl* que implementa parcialmente esta interfaz y se espera que la mayoría sino es

que todos los modelos extiendan esta clase. Estas entidades se muestran en la Figura 29 dentro del paquete *uchicago.src.sim* que forma parte del API de Repast. Dentro de este mismo paquete se encuentra la interfaz *Drawable* que debe implementarse cuando los agentes se despliegan visualmente.

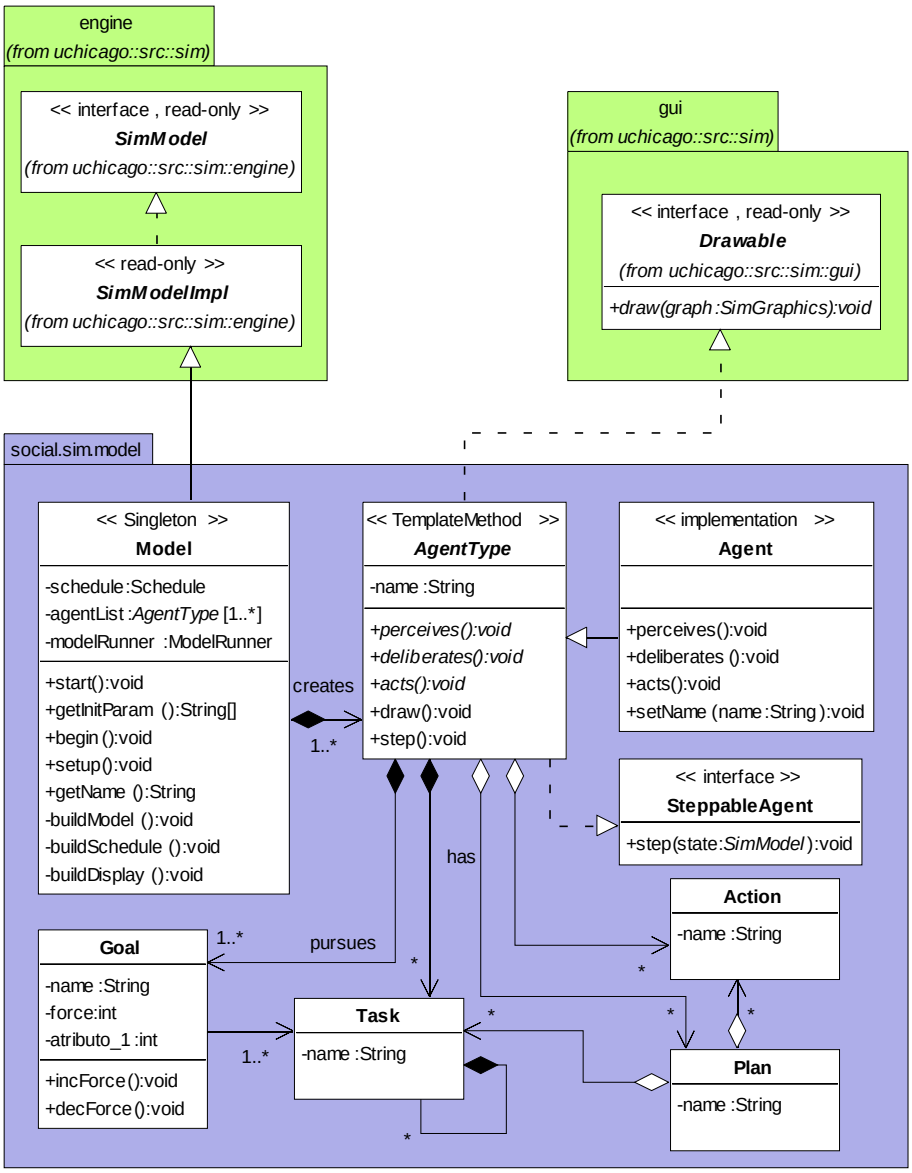

Figura 29. Diagrama de Clases del Prototipo de Agente y de Modelo SSBA con Repast.

En la misma Figura 29 se observa el paquete *social.sim.model* que contiene las entidades de implementación para el prototipo de modelo y agente siguiendo el modelo computacional para la SSBA. Por un lado se encuentra la clase *Model* que configura los parámetros y crea las variables

de infraestructura como el *planificador Schedule* y las entidades de representación como los agentes. Así, la clase modelo contiene una lista de todos los agentes de una simulación y los referencia vía la clase abstracta *AgentType*. Esta clase representa un tipo de agente y proporciona un punto de acceso estándar para referenciar a los agentes dentro del modelo, el método *step* de todos los agentes en una simulación es llamado en cada paso de la simulación. Por lo tanto, esta clase debe implementar también la interfaz *SteppableAgent* que contiene el método *step* con el que todos los agentes deben contar. La implementación de este método que sigue el patrón de diseño *template method* (véase sección 5.2) invoca los métodos que definen el comportamiento del agente (*perceives*, *deliberates* y *acts*). Como podemos ver en la Figura 29 esta interfaz forma parte del paquete *social.sim.model* y es la propuesta que hacemos para fomentar que sean los agentes los que sean *programados* para ejecución y no sus acciones individuales como se hace en Repast. La clase *AgentType* también mantiene relaciones con las clases que representan los objetivos del agente *Goal*, sus planes *Plan*, las tareas *Task* y las acciones *Action.* En la clase para los objetivos podemos observar el valor de motivación que se les ha añadido para la implementación del mecanismo de adaptabilidad por refuerzo.

El prototipo del entorno se implementa con las entidades de la Figura 30. El entorno es simulado con alguna de las clases del paquete *uchicago.src.sim.sapce* la cual permite que los agentes esten inmersos en relaciones de espacio unos con otros. Aunque este espacio no es necesario en todas las simulaciones, ya que cabe la posibilidad de implementar un entorno sin relaciones espaciales en esta propuesta. La interfaz *EnvironmentInterface* y la clase concreta *Environment* pertenecientes al paquete proporcionado para la SSBA *social.sim.model* dotan de flexibilidad al prototipo del entorno. La interfaz proporciona un punto de acceso estandar para todas las implementaciones de entorno por medio de operaciones comunes, la clase concreta implementa esta interfaz con un objeto espacio *Object2DGrid* de Repast, el cual es un objeto que representa un espacio en forma de rejilla en dos dimensiones. Esta clase concreta *Environment* sigue el patrón de diseño *adapter* (véase sección 5.2) y permite reemplazar el objeto espacio fácilmente por otro cualquiera.

Por último, un componente importante para una simulación es el *planificador*. En Repast este mecanismo es implementado con la clase *Schedule*. Este consiste en configurar llamadas a métodos en objetos para que ocurran en cierto momento. Estos métodos deben ser "envueltos" por una subclase de la clase *BasicAction*. Esta clase tiene algunas variables utilizadas por el *planificador Schedule* y un método abstracto *execute*. Es en este método donde las llamadas a los métodos que han de programarse en Repast deben ocurrir. Son estas dos clases los principales componentes del mecanismo de planificación de Repast y pueden observarse en la Figura 31 en el paquete *uchicago.src.sim.engine*.

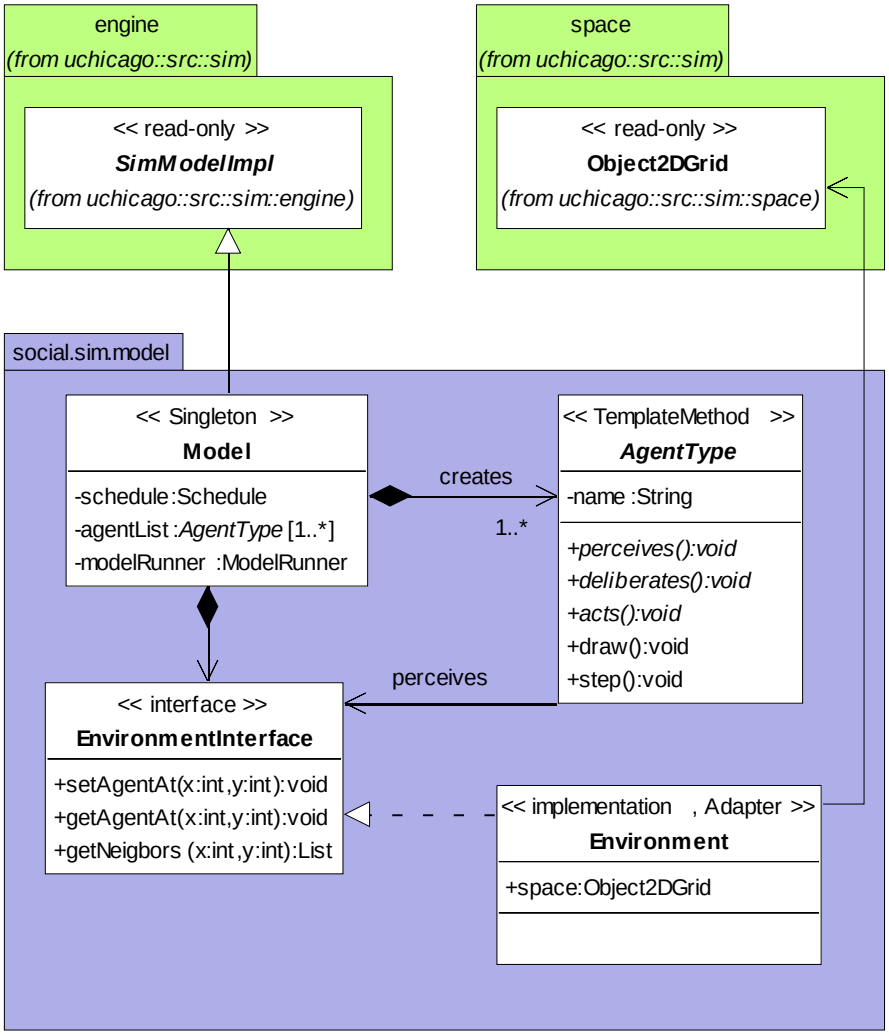

Figura 30. Diagrama de Clases del Prototipo del Entorno SSBA con Repast.

El mecanismo de planificación de Repast, aunque es bastante flexible, no se ajusta a algunas de las ideas que motivan una propuesta basada en SMA, ya que el crear una lista de acciones para ser tomadas en cada paso de tiempo, se asemeja más a una propuesta de coordinación central. Sin embargo, existe cierta flexibilidad en el API y se puede evitar este funcionamiento. Por esta razón se ha realizado un diseño que permite programar la ejecución de un agente como una entidad autónoma y no sus acciones individuales. De esta forma los agentes actúan en cada paso de tiempo y son ellos los que deciden que acciones ejecutar. Las entidades que permiten esta funcionalidad pueden observarse en la Figura 31 en el paquete *social.sim.model*, también puede observarse su relación con las entidades del mecanismo planificador de Repast.

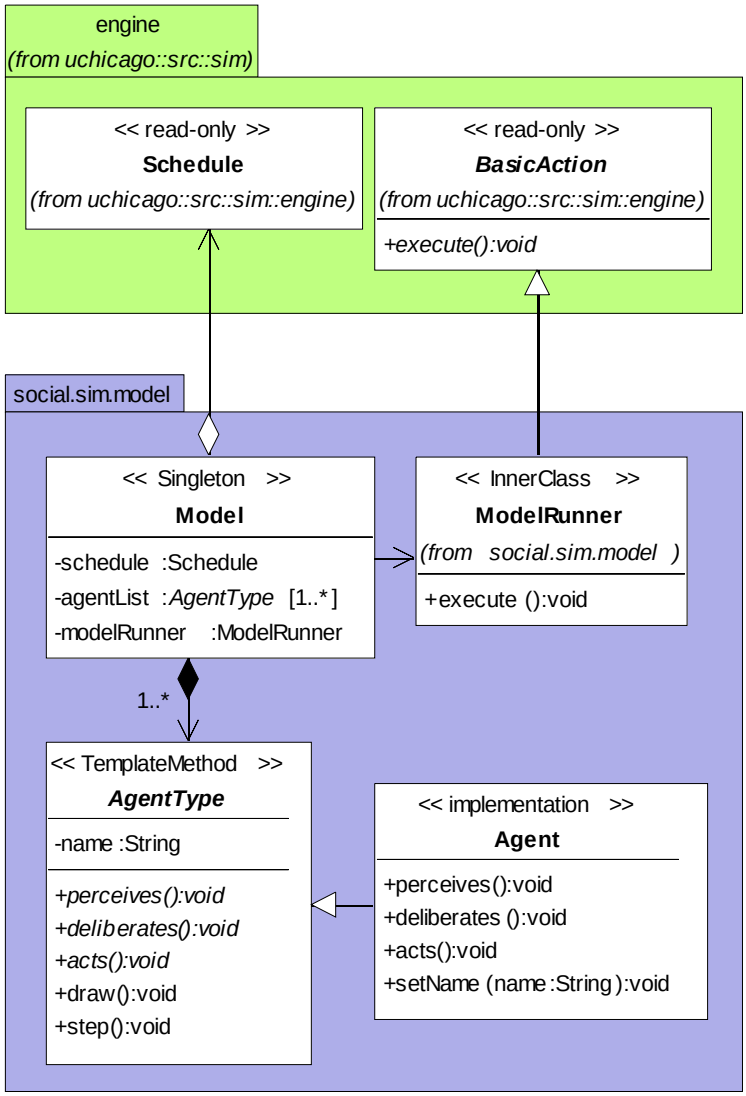

Figura 31. Diagrama de Clases para el la Planificación del Modelo SSBA con Repast.

En la Figura 31 podemos observar que la clase *Model* contiene una clase interna (*inner class*) llamada *ModelRunner* la cual es una subclase de la clase *BasicAction* de Repast. De esta forma, la clase *ModelRunner* es programada para su ejecución en cada paso de la simulación ejecutando el método *execute*, y es en este método donde se permite a cada agente de la simulación ejecutarse. Esto quiere decir que es en la estructura interna de cada agente donde se incluye el mecanismo de decisión de qué acciones ejecutar en cada paso.

5.5. Conclusiones

Los mecanismos y herramientas presentados en este capítulo facilitan la aplicación del marco metodológico para el desarrollo de modelos de SSBA. Siguiendo la línea de razonamiento planteada en la arquitectura del marco metodológico (véase la sección 6.2), las herramientas revisadas facilitan las diversas etapas de la ingeniería de desarrollo MDE. Específicamente, las herramientas se concentran en la transformación de los modelos para la generación automática de código y la validación de los modelos.

Para automatizar la implementación de los modelos de simulación se presentó el módulo para la generación de código el cual se basa en la definición de un conjunto de prototipos que componen una simulación, plantillas y un algoritmo de recorrido de la especificación del sistema. Como ejemplo de lo anterior se presentó el módulo desarrollado para Repast.

Para la validación fue necesario considerar dos aspectos, uno que los modelos de ejecución fueran funcionalmente equivalentes a los modelos conceptuales, y otro que se preservara la semántica de los elementos del modelo abstracto de agentes con los de ejecución. Para realizar estos aspectos fue necesario incluir un modelo computacional de SMA a las plataformas de simulación ya que el modelo de éstas es más limitado que el de INGENIAS que considera aspectos de intencionalidad y racionalidad de los agentes en línea con (Newell 1982).

También se definió un mecanismo de adaptabilidad por refuerzo para modelar agentes sociales o sociedades con características de auto-organización. Este mecanismo es flexible y reutilizables pues las reglas con las cuales funciona pueden redefinirse.

Algunas de las ventajas que representa la aplicación de estas herramientas en el dominio de la SSBA son:

1. Reducen la necesidad de conocimiento de programación a nivel experto por parte del usuario del dominio.
2. Permiten a los expertos del dominio enfocar su esfuerzo en el modelado y evaluación de resultados.
3. Flexibilidad en la utilización de plataformas de simulación.
4. Permiten la implementación de SA con un modelo más desarrollado de SMA.
5. Facilitan la implementación de agentes cognitivos.
6. Facilitan la verificación de errores difíciles de observar gracias a la evolución iterativa y en paralelo del desarrollo del modelo y la implementación con contra-verificaciones constantes durante la implementación del módulo para la generación de código de simulación.
7. Permiten ir validando internamente el modelo incrementalmente, es decir, el desarrollador del sistema puede generar instancias o prototipos tomados de modelos intermedios para probar su funcionalidad antes de que el sistema sea implementado completamente.

8. Permiten la interpretación de los resultados de la simulación en términos del modelo.
9. Facilitan la replicación pues permiten ejecutar las especificaciones en distintas plataformas para replicar y comprobar resultados. También permiten estudiar los efectos que pueden tener las facilidades de distintos entornos de simulación sobre los resultados de la simulación.

Finalmente, para incluir nuevos entornos de simulación diferentes a los que se revisaron en esta propuesta hace falta la intervención de un ingeniero informático que sepa utilizar la interfaz de programación del IDK. En principio esto podría parecer una limitación. Sin embargo, esto implica un sólo esfuerzo de desarrollo, pues una vez que el módulo correspondiente se ha desarrollado las tareas repetitivas y complejas de la plataforma de simulación quedan resueltas en el módulo. Por lo tanto, la utilización de los prototipos proporcionados por éste permite agilizar el desarrollo de futuras simulaciones.

Capítulo 6.

Marco Metodológico para el Estudio de Sociedades Artificiales

En este capítulo se describe el marco metodológico para el estudio de SA. El marco comprende herramientas de modelado y desarrollo y una metodología para la ingeniería y estudio de los modelos de simulación. En los capítulos anteriores se describió el lenguaje de modelado y las herramientas. En este capítulo se describe particularmente la metodología que éste plantea y los principios en las cuales se cimienta. Además, se detalla cómo se integran las herramientas en el proceso metodológico indicando sus funciones en cada paso. Para la ingeniería, consideramos primordial tomar como referencia una metodología de la ingeniería del software orientada a agentes (Jennings 2000), concretamente la metodología INGENIAS (Pavón et al. 2002) cuyo planteamiento está en línea con la ingeniería de desarrollo basada en modelos (Model Driven Engineering, MDE). Para el estudio, se define un proceso que facilita la experimentación e interpretación de los modelos. Ambos procesos se integran para proporcionar una metodología ágil para el estudio de SA.

6.1. Introducción

La necesidad de una metodología para el estudio de SA se fundamenta en el análisis del estado del arte en el capítulo 3. Tal como se ha visto, las metodologías evaluadas se concentran en las etapas de experimentación y eliden el desarrollo de SMA con los que se modelan las SA. Algunas metodologías que si consideran esta fase se basan en modelos de agentes sencillos, con lo cual el proceso de elaboración que proponen es incompleto cuando se trata de modelar sociedades de agentes inteligentes. Otras más avanzadas como (Drogoul et al. 2002) consideran la necesidad de un modelo de SMA más desarrollado, sin embargo no define cómo ha de desarrollarse.

Así, la metodología propuesta para abordar esta problemática proporciona una guía de desarrollo de modelos de SMA que considera aspectos de intencionalidad y racionalidad de los agentes. Esto implica definir un proceso bien definido para la elaboración de su diseño, y la transformación de su especificación a un modelo de simulación ejecutable. Por otro lado, este proceso de desarrollo se ubica en un proceso más general de estudio que planteamos en línea con (Gilbert y Troitzsch 1999; Gilbert 1999; Ramanath y Gilbert 2003; Axelrod 1997a; Axtell et al. 1996). El propósito de este proceso de estudio es identificar cómo ciertas etapas de la investigación pueden beneficiarse de la propuesta y herramientas de desarrollo del marco metodológico, concretamente la experimentación, la interpretación, la replicación y el análisis de patrones sociales.

Para facilitar la ingeniería de modelos de SA con agentes inteligentes se ha tomado como referencia la metodología INGENIAS. Con ésta definimos un ciclo de vida para el desarrollo de estos modelos adaptando ciertos aspectos específicos al dominio del modelado de sistemas sociales. Estas adaptaciones se reflejan en las herramientas de modelado y generación de código que se revisaron anteriormente (véase los capítulos 4 y 5) pues son éstas las que impulsan la adopción del proceso metodológico durante el estudio. Sin embargo, algunas adaptaciones que se realizaron serán más evidentes durante la descripción del proceso de estudio aquí propuesto, por ejemplo las adaptaciones para extender las perspectivas del modelo de SMA de INGENIAS para SA.

La ingeniería de modelos SA propuesta se basa en las tecnologías que plantea MDE para el desarrollo de aplicaciones (Beydeda 2005). Fundamentalmente, MDE trata de elevar el nivel de abstracción de la descripción del problema a resolver y de su posible solución, y establecer un proceso para ir del nivel de descripción al nivel de la solución. Para tratar el nivel de abstracción proponemos la especificación de los modelos con el lenguaje de modelado LMSA, para abordar la solución del problema proponemos la realización de los modelos de simulación con el módulo de generación de código, y el proceso para ir del primero al segundo lo define la metodología de estudio que proponemos.

La sección 6.2 introduce los principios de la metodología. La sección 6.3 presenta los componentes del marco metodológico. La sección 6.4 define el proceso metodológico, junto con los roles que participan en éste. La sección 6.5 discute las ventajas de esta propuesta metodológica.

6.2. Principios de la Metodología

La abstracción es el principio más básico de la ingeniería del software. Las abstracciones son proporcionadas por *modelos*. El *modelado* y las *transformaciones de modelos* constituyen la naturaleza de la ingeniería MDE. Los modelos pueden refinarse y finalmente ser transformados en una implementación técnica, es decir, un sistema de software. Así, bajo esta perspectiva, el proceso de desarrollo se concibe como un proceso de transformación de modelos abstractos, independientes de detalles de la plataforma, a código fuente o modelos de ejecución.

Como el proceso metodológico que se propone se basa en estos principios se considera útil una introducción a la filosofía MDE para el desarrollo del software.

6.2.1. Ingeniería de Desarrollo Basada en Modelos

La ingeniería de desarrollo basada en modelos (*Model Driven Engineering*, MDE) se basa en los principios de 1) separar la descripción de las propiedades abstractas y la lógica de una aplicación de la descripción de su implementación en una plataforma específica, y en 2) automatizar la transformación de la primera en la segunda usando herramientas avanzadas de transformación de modelos.

Los modelos proporcionan abstracciones de un sistema físico que permiten a los ingenieros entender ese sistema ignorando detalles superfluos, enfocándose en los relevantes. Todas las formas de ingeniería dependen de modelos como algo imprescindible para comprender sistemas complejos del mundo real. Los modelos pueden desarrollarse como precursor de un sistema físico, o pueden derivarse de un sistema existente o un sistema en desarrollo, como ayuda para comprender su comportamiento.

Hay muchos aspectos de un sistema que pueden ser de interés. Dependiendo de lo que es considerado relevante en cierto momento pueden usarse varios conceptos y notaciones de modelado que destacan una o más perspectivas particulares de ese sistema. Aún más, en algunos casos, los modelos pueden aumentarse con información o reglas que asisten en su transformación. Por ejemplo, con frecuencia es necesario convertir entre diferentes *perspectivas* del sistema a un nivel equivalente de abstracción (p.ej. entre una perspectiva estructural y una perspectiva de comportamiento), y las transformaciones de modelos facilitan esta tarea. En otros casos, una transformación puede

definirse entre distintos niveles de abstracción, generalmente, de una perspectiva más abstracta a una menos abstracta, añadiendo más detalles proporcionados por las reglas de transformación.

Actualmente, la formulación más consensuada de esta visión es la de OMG (*Object Management Group*) que propone la arquitectura MDA (*Model Driven Arquitecture*) (OMG 2003a) en forma de un conjunto de estándares para integrar las herramientas que soporten la MDE. Las descripciones independientes de la plataforma pueden construirse utilizando estándares de modelado de OMG como UML (*Unified Modeling Language*) (OMG 2003c), MOF (*Meta-Object Facility*) (OMG 2003b), y CWM (*Common Warehouse Metamodel*) (CWM 2001), y pueden transformarse a plataformas propietarias como CORBA, J2EE, .NET, XMI/XML, y plataformas Web. Para las transformaciones, OMG tiene la intención de adoptar el lenguaje QTV (Query/View/Transformation) (OMG 2007) que deberá permitir hacer consultas sobre los modelos MOF, de crear diferentes perspectivas de un modelo y de definir las transformaciones. También existe otra serie de estándares para el intercambio de modelos, para el almacenamiento y evolución de los modelos, y recientemente para la generación de código.

Para soportar esta propuesta OMG ha definido un conjunto de niveles y transformaciones que proporcionan un marco conceptual y un vocabulario para MDA. En concreto, se identifican cuatro niveles: Modelo Independiente de Computación (*Computation Independent Model*, CIM), Modelo Independiente de la Plataforma (*Platform Independent Model*, PIM), Modelo Especifico de la Plataforma (*Platform Specific Model*, PSM) descrito por un Modelo de Plataforma (*Platform Model*, PM), y un Modelo Específico de Implementación (*Implementation Specific Model*, ISM). Estos niveles se ilustran en la Figura 32.

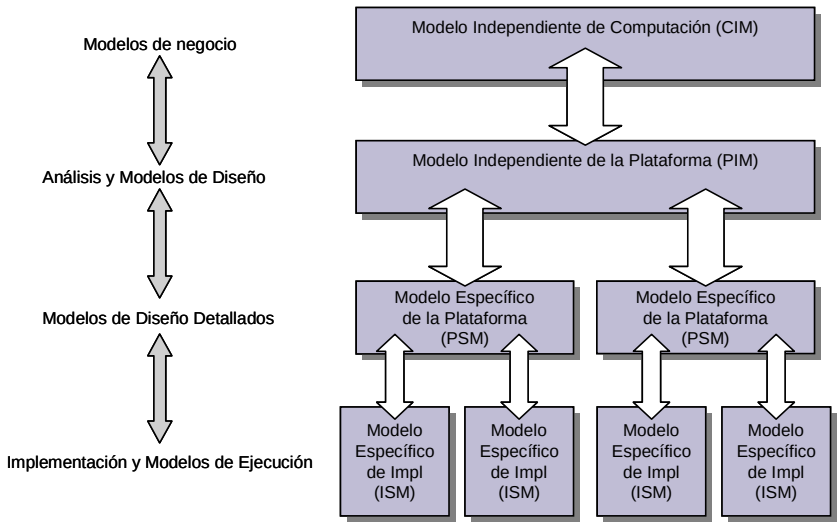

Figura 32. Niveles y sus Respectivos Modelos en la Arquitectura MDA (adaptado de (OMG 2003a))

La arquitectura MDA se basa en la utilización de modelos independientes de la plataforma (PIM), que ayudan a desarrollar el software en un nivel más elevado de abstracción, ocultando detalles específicos de la plataforma. Así, éstos resuelven algunos de los problemas que son causados por la creciente complejidad de los sistemas software. Para una mayor reducción de la complejidad, los PIMs pueden modelarse con varias *perspectivas*, esto último con el propósito de enfocarse en aspectos particulares dentro del sistema de forma separada. Para ejecutar el PIM en una plataforma destino, debe generarse un modelo específico de la plataforma (PSM).

MDA también se basa en las transformaciones de modelos. Las transformaciones pueden ser esencialmente de dos tipos: 1) transformaciones PIM-a-PIM o *transformaciones horizontales* y 2) transformaciones PIM-a-PSM o *transformaciones verticales*. En el primero los modelos se transforman en otros modelos en el mismo nivel de abstracción, por ejemplo cuando se migra la especificación de un modelo a otro lenguaje o cuando se construye un modelo con perspectivas nuevas basadas en otras existentes. En el segundo, los modelos se transforman en otros modelos en un nivel de abstracción diferente. Por ejemplo, en un nivel inferior cuando se trata de sintetizar o generar código, superior cuando se trata de ingeniería inversa. En la Figura 33 podemos observar la relación de estos componentes o principios fundamentales de la MDA cuando se aplican a una transformación vertical.

Es interesante notar que el concepto de "plataforma" es complejo, y en gran medida dependiente del contexto. Por ejemplo, en algunas situaciones la plataforma puede ser el sistema operativo y las utilerías asociadas, y en otras puede ser la infraestructura tecnológica representada por un modelo de programación bien definido como J2EE o .NET. Lo más importante, independientemente de lo que consideremos como una plataforma, es "*pensar en términos de modelos en diferentes niveles de abstracción usados para diferentes propósitos*". Con lo anterior queremos decir que los modelos que no se encuentren en el nivel más alto o más bajo de abstracción juegan el papel de PIMs en primer lugar y de PSMs posteriormente. Por ejemplo, un modelo que ha sido refinado es llamado PSM, y el modelo en el nivel de abstracción inmediatamente arriba de éste, el cual fue la fuente para el refinamiento, es llamado PIM. Si este PSM no se encuentra en el nivel más bajo de abstracción, entonces juega el papel de PIM para el siguiente modelo que se derive de éste, y así sucesivamente. Por lo tanto cada PSM es relativo a cada PIM y viceversa. Finalmente, a partir del PSM de más bajo nivel se puede obtener el modelo específico de implementación, ISM (véase la Figura 32), el cual es una descripción del sistema a nivel de código.

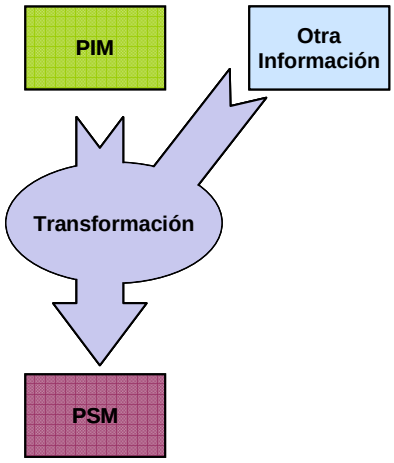

Figura 33. Principios Fundamentales de la Arquitectura MDA (OMG 2003a)

En base a los principios de la ingeniería MDE anteriormente revisados, se establece una serie de requisitos que han de cumplirse para poder definir un proceso MDE:

1. Primero se han de diseñar los modelos. Para esto, debe especificarse la secuencia de modelos que han de desarrollarse (cuantos niveles de abstracción hay), qué plataformas deben integrarse, y cuál es la notación de modelado y la sintaxis abstracta en cada nivel de abstracción.
2. También ha de señalarse cómo derivar un modelo de otro que se encuentra en el nivel de abstracción inmediatamente superior y qué información debe añadírsele. El sistema es descrito por un modelo en un nivel de abstracción muy alto. Este modelo es conocido como CIM en la terminología de MDA. Éste no incluye ningún detalle de la estructura del sistema y suele llamarse también Modelo del Dominio, con lo cual se usa un vocabulario familiar a ese dominio para su especificación. Los *casos de uso* son usados muchas veces para desempeñar el papel del CIM. Este modelo se irá transformando progresivamente para hacer el sistema más específico de la plataforma. Por ejemplo, el sistema puede ser expresado ahora con más precisión con un *diagrama de clases*. Más adelante, se puede añadir información adicional para integrar aspectos de distribución, y a este modelo de distribución añadirle posteriormente información cada vez más específica (siguiendo la terminología de MDA este sería el modelo PSM).
3. Se ha de indicar cómo generar código para el lenguaje de modelado usado en el nivel más bajo de abstracción, lo que podríamos llamar el modelo ISM en la terminología MDA.
4. También debe definirse cómo verificar un modelo con respecto al modelo más abstracto, cómo validarlo, y cómo podrían generarse pruebas del sistema que se está desarrollando. En cada paso del proceso MDE se puede añadir información relacionada con estos aspec-

tos. Una verificación puede ser la acción que controla si un PSM no rompe con la especificación promovida por su PIM, o viceversa (en el caso de la ingeniería inversa). El paso de validación debe permitir al desarrollador del sistema generar instancias o prototipos tomados de modelos intermedios para probar su funcionalidad antes de que el sistema sea implementado completamente.

6.3. Componentes del Marco Metodológico

Integrando los conceptos, tecnologías y herramientas anteriormente revisadas se define el marco metodológico para el estudio de SA. Los componentes del marco pueden observarse en la Figura 34. En el lado izquierdo de la figura se encuentran los mecanismos y herramientas que soportan el marco metodológico. Entre éstas el lenguaje de modelado, el mecanismo de transformación, el modelo computacional SMA, las plataformas de simulación, etc. En el centro se encuentra el proceso para el desarrollo de modelos de SA, y en el lado derecho se indica la correspondencia con el proceso MDE.

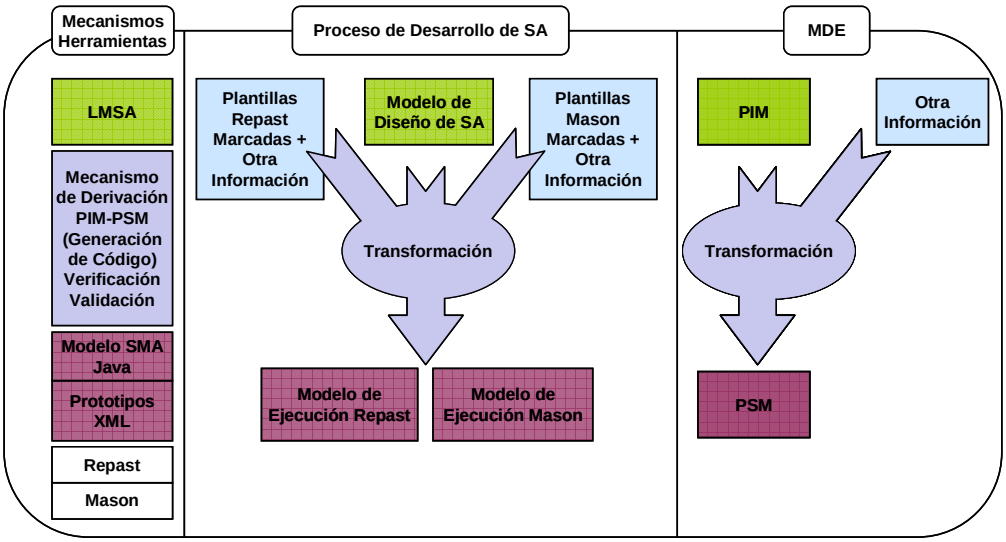

Figura 34. Componentes del Marco Metodológico para el Desarrollo de Modelos de SA

El marco metodológico sigue los principios MDE, con la definición de los siguientes aspectos y la participación de las siguientes herramientas:

1. Se distinguen dos niveles de abstracción: el modelo más abstracto o *modelo de diseño* del SMA (PIM) y el *modelo de ejecución* (PSM). En el nivel de abstracción más alto utilizaremos la notación de modelado proporcionada por la sintaxis del lenguaje LMSA, y su correspondiente sintaxis abstracta es representada por los meta-modelos de INGENIAS y sus

relaciones. En el nivel de abstracción más bajo, la sintaxis notacional para los modelos es la de los lenguajes de simulación Repast y Mason cuya sintaxis abstracta es la del lenguaje Java, con el cual están desarrolladas ambas plataformas.

2. La derivación del *modelo de diseño* al *modelo de ejecución* sobre las plataformas Repast y Mason es un proceso iterativo. En la sección 5.4 se explica cómo realizar esta derivación con el módulo para la generación automática de código.
3. Como solo se proponen dos niveles de abstracción o dos modelos, el modelo menos abstracto que proponemos es el que llamamos *modelo de ejecución*. Éste es un modelo PSM porque cuenta con características particulares de la plataforma de simulación, pero a la vez cumple la función de un modelo ISM de implementación ya que es generado como código fuente para las plataformas de simulación. Por lo tanto, la generación de código se lleva a cabo con el mecanismo de derivación mencionado anteriormente en el inciso 2.
4. Por último, la validación y verificación del modelo PSM con su respectivo PIM se propone como sigue. La verificación se lleva a cabo con la generación de código, durante la cual se garantiza que el PSM sigue la especificación promovida por el PIM. Esta verificación es facilitada por el nivel de abstracción para la implementación de SMA que hemos añadido en el PSM. Los conceptos de esta abstracción computacional corresponden a los del lenguaje LMSA directamente. Aparte de esta verificación también es posible a través del IDK de INGENIAS llevar a cabo la verificación de propiedades del modelo PIM. Esto se realiza por medio de un módulo de verificación que analiza si el modelo cumple un conjunto de requisitos, recorriendo el modelo y analizando la satisfacción de las propiedades para los que hayan sido diseñados. Esta verificación no es directa, primero habrían de definirse los requisitos a cumplir. En lo que respecta a la validación, siempre es posible generar modelos de ejecución incrementalmente, lo que le permite a los expertos del dominio probar distintas funcionalidades dinámicamente.

La Figura 35 muestra cual es la funcionalidad de los componentes del marco metodológico. En la figura puede observarse que en el nivel de abstracción más alto se encuentra la especificación del modelo de diseño con el lenguaje LMSA, el proceso de generación de código se encuentra entre el nivel más alto y el nivel más bajo de abstracción. En este último nivel se encuentra el modelo computacional de SMA con agentes intencionales sobre las plataformas Repast y Mason.

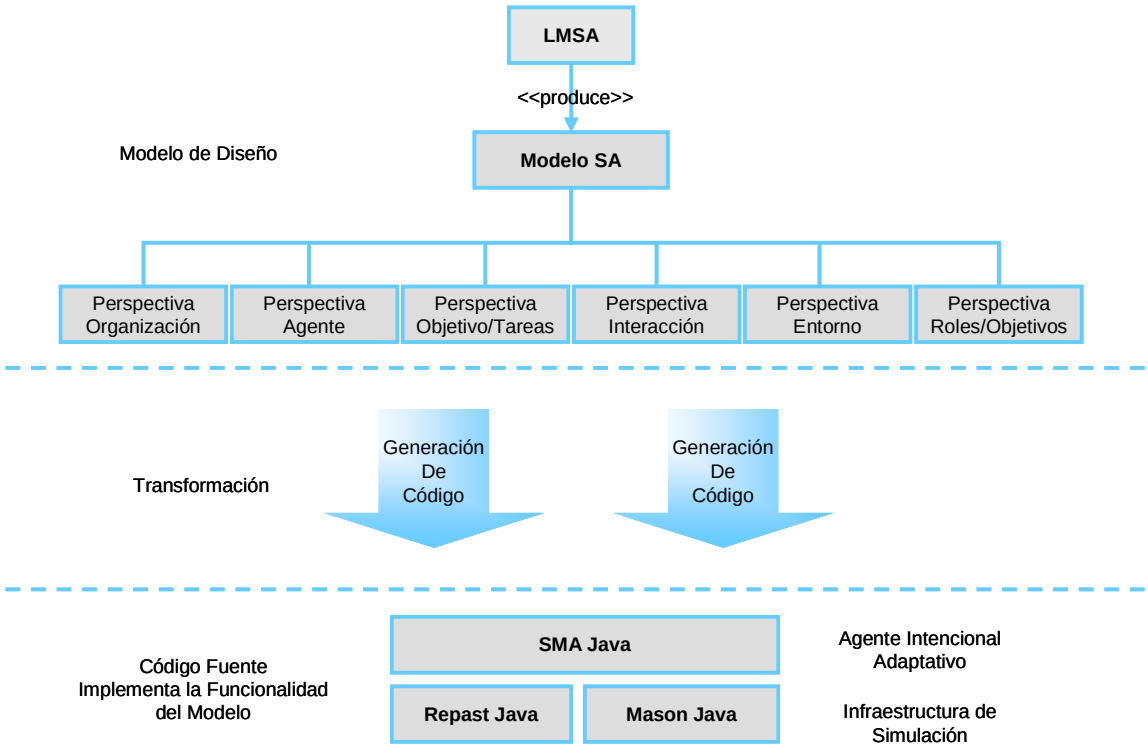

Figura 35. Interacción y Funcionalidad de los Componentes del Marco Metodológico para el Desarrollo de SA

6.4. El Proceso Metodológico

En términos generales, un estudio de simulación se inicia con la *fase de identificación*, es decir, *la definición del problema* con respecto a un fenómeno en un dominio particular. Con la *formulación del problema* se identifican los elementos del sistema y sus relaciones. La *fase de ingeniería* es la fase inmediata a la de identificación. Se trata de un proceso iterativo en el cual el conocimiento informal del sistema identificado es transformado en un modelo más formal y en última instancia en una implementación o programa ejecutable.

La relación entre estas dos fases es a través de las entidades que se muestran en la Figura 36, el *fenómeno real*, el *sistema*, el *modelo*, el *formalismo* y la *implementación*. El sistema se define a partir de los actores, relaciones y comportamiento de un fenómeno real. El modelo es la especificación del sistema mediante un formalismo o lenguaje que define la sintaxis y la semántica del modelo. Los modelos basados en este lenguaje son restringidos por las reglas de su semántica. La implementación se genera mediante la codificación del modelo formal en un programa ejecutable.

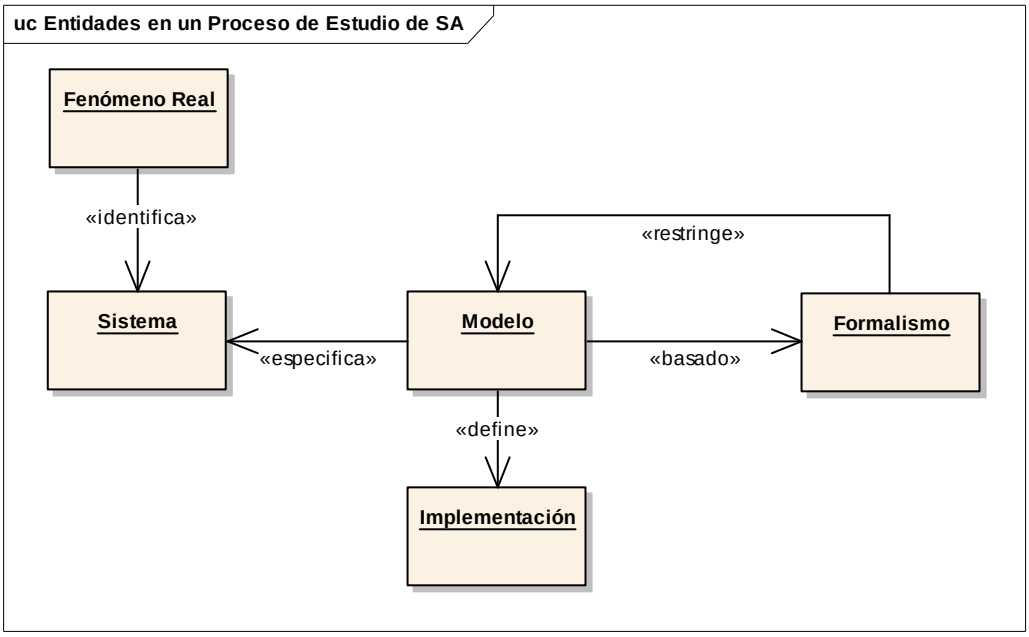

Figura 36. Entidades en un Proceso de Estudio de SA

En la propuesta del marco metodológico, la función del *formalismo* la realiza el lenguaje LMSA y la implementación a través del mecanismo de transformación de los modelos abstractos a modelos de implementación.

Por otro lado, durante el proceso de estudio de SA definido por el marco metodológico se identifican dos roles:

- *Rol experto del dominio*. En un proceso de simulación social este rol se encarga de definir el sistema objetivo. Este sistema describe el fenómeno que se ha de predecir o la teoría que necesita explicación. Este rol involucra expertos en un dominio particular o en un tema específico, por ejemplo un sociólogo, el cual definirá el propósito del proceso de simulación. El conocimiento con el que ha de contar el participante que desempeñe este rol es el *macro conocimiento* del sistema objetivo y el *micro conocimiento* que es el conocimiento local que se tiene de los *individuos*. Este rol suele producir el *modelo del dominio*, que contiene "agentes" y comportamiento definido usando algún lenguaje específico del dominio o generalmente mediante descripciones en lenguaje natural. El modelo producido suele ser ambiguo por lo cual se requieren otros modelos para poder construir el programa de simulación.

  En esta propuesta se sugiere usar directamente el lenguaje LMSA. Así, este rol producirá directamente el *modelo de diseño* basándose en los conceptos de agentes y SMA del lenguaje LMSA.

- *Rol ingeniero software*. Es responsable de definir el mecanismo de generación de código automático para las simulaciones, por lo tanto, se puede decir que genera los *modelos específicos de ejecución o computacionales*. Se encarga también de implementar el lenguaje LMSA personalizando el IDK al dominio social. Cuenta con conocimientos del IDK, meta-modelado, y entornos de simulación.

La colaboración de estos dos roles puede observarse en la Figura 37. Estas colaboraciones se realizan antes de que el proceso de estudio de SA pueda llevarse a cabo, pues con éstas se adquiere el conocimiento necesario para definir el lenguaje de modelado adecuado al dominio de aplicación y el módulo para la generación de código. En la Figura 37 se muestra solamente la colaboración de estos roles, en el capítulo 4 y 5 se describe el proceso completo para la elaboración de estas herramientas y la participación de ambos roles en las actividades de este proceso. En la misma figura se puede observar que el rol experto proporciona el conocimiento del dominio social y el rol ingeniero software el conocimiento para extender los meta-modelos de INGENIAS y para generar código para las plataformas de simulación.

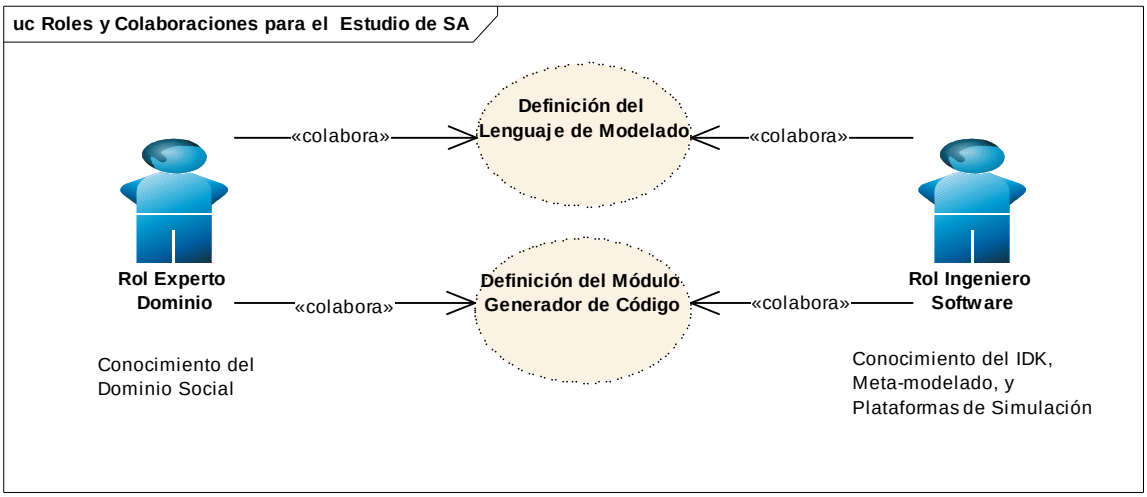

Figura 37. Roles y Colaboraciones para el Estudio de SA

La interacción y colaboración que realizan estos roles durante el estudio de SA pueden observarse en la definición de este proceso metodológico, concretamente en cada una de las fases y actividades que éste propone.

Finalmente, ambos roles se pueden refinar en roles más especializados. Por ejemplo, el rol experto del dominio se podría especializar en un rol modelador, y el rol ingeniero software en rol analista, diseñador, programador, ingeniero de pruebas, ingeniero de calidad, etc.

6.4.1. Fases y Actividades en el Proceso

El proceso metodológico para el estudio de SA es un proceso iterativo e incremental pues realiza constantes verificaciones con las distintas fases que lo componen para ajustar, redefinir o incrementar el modelo, ya sea por errores que se hayan cometido durante las fases anteriores o por mejorarlas mediante retroalimentaciones. La Figura 38 esquematiza la filosofía del proceso metodológico SA.

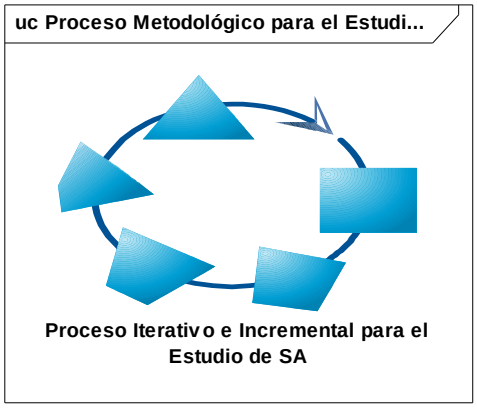

Figura 38. Proceso Metodológico para el Estudio de SA

El proceso metodológico está divido en tres fases, como se muestra en la Figura 39. La primera, la *fase de identificación del objeto de estudio*, y la segunda, la *fase de ingeniería de SA*, son ambas parte del proceso de desarrollo. La tercera fase, la *fase de experimentación*, completa el ciclo de un proceso de investigación, el cual incluimos para identificar cómo parte de la funcionalidad del marco metodológico facilita igualmente el proceso de investigación. Las actividades que se realizan en cada fase se han agrupado en procesos más grandes representados como *flujos de trabajos*.

La *fase de identificación del objeto de estudio* se compone de la *definición del sistema*. Seguidamente, la *fase de ingeniería de SA* implica la realización del sistema identificado, ello implica la *generación del modelo conceptual* y *la generación del modelo específico de la plataforma*. La elaboración de estos dos últimos procesos también se lleva a cabo con constantes contraverificaciones. Finalmente, la *fase de experimentación* implica realizar el proceso para el *estudio de propiedades emergentes*. Estos flujos de trabajo pueden observarse en la Figura 39.

A continuación se describen las actividades y los roles participantes en cada uno de los flujos de trabajo que conforman las fases del proceso metodológico.

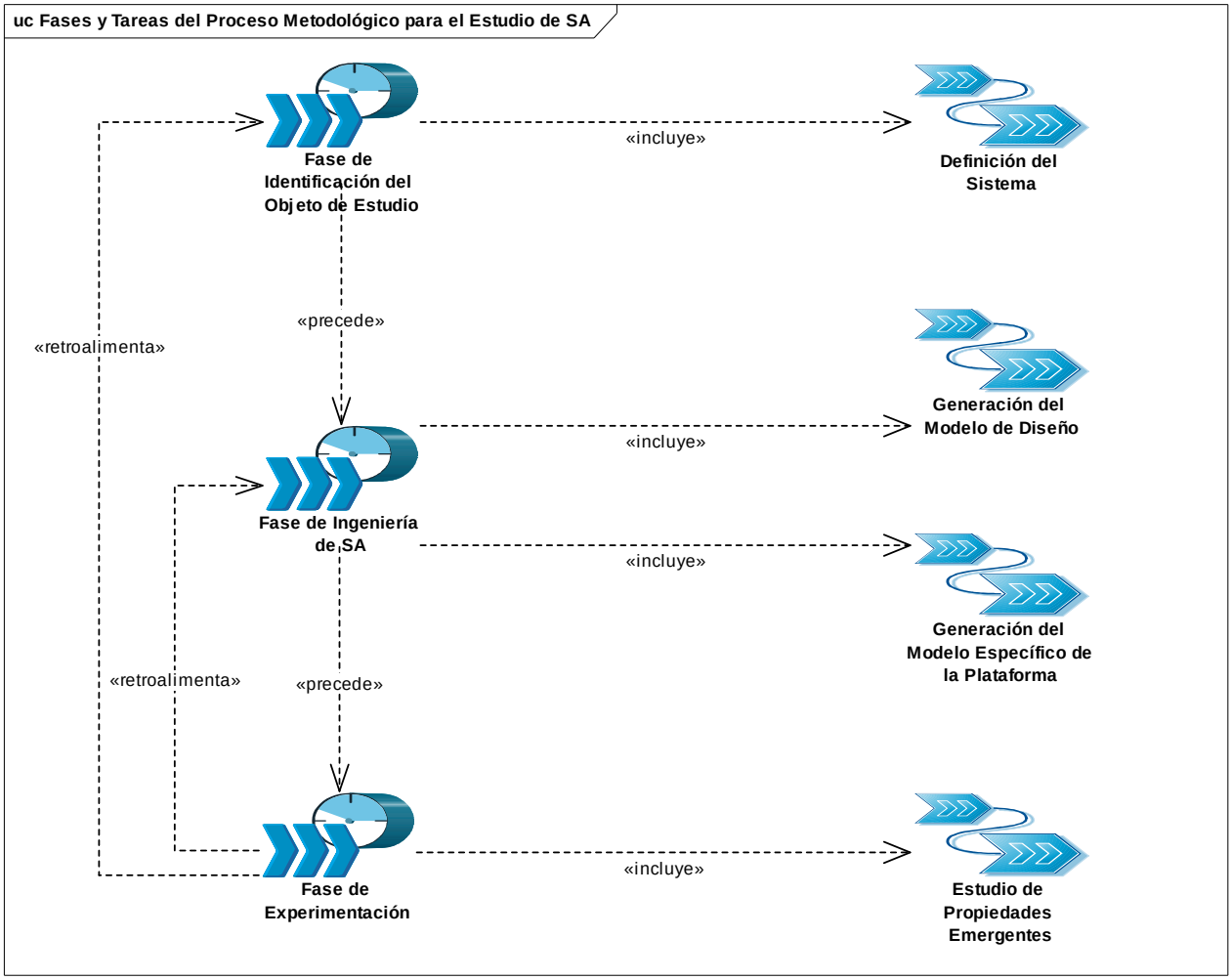

Figura 39. Fases y Flujos de Trabajo del Proceso Metodológico para el Estudio de SA

6.4.1.1.Fase de Identificación del Objeto de Estudio

El proceso de desarrollo de un estudio de simulación se inicia con esta fase cuando el experto del dominio social busca una explicación a alguna teoría o busca comprender algún fenómeno específico. Para ello, ha de definir y formular el problema en relación al fenómeno o teoría bajo estudio.

Flujo de Trabajo: Definición del Sistema

Roles: Experto del Dominio

Entradas: Sistema Real

Salidas: Sistema y sus Objetivos, Conocimiento Micro y Macro

Lenguajes y Herramientas: Texto

Actividades:

1. *Definición del problema*. El experto adquiere conocimiento del objeto de estudio por medio de observaciones que ha llevado a cabo, realiza teorías y asunciones sobre este mismo y se hace preguntas que busca resolver por medio de la simulación. Las observaciones son datos relativos al fenómeno, que pueden usarse como parámetros o como condiciones iniciales de la simulación. Las preguntas que formula pueden ser: de predicción, especulativas o teóricas.
2. *Formulación del problema*. Una vez definido el problema bajo estudio se pasa a la formulación del mismo. En este paso se identifica el sistema dentro del dominio de aplicación social. El experto formula los objetivos en relación a las preguntas que el sistema ha de responder con respecto al fenómeno o teoría. En este paso también puede organizar el conocimiento del sistema que ha adquirido o con el que ya contaba, en dos niveles, el *macro conocimiento* que es el conocimiento global que se tiene del sistema obtenido en su mayoría de las observaciones y el *micro conocimiento* que es el conocimiento local que se tiene de los "*individuos"* obtenido también de las observaciones (comportamiento, etc.) y de las asunciones. Es aquí donde la SSBA tiene el potencial de enlazar estos dos niveles, permitiendo a los expertos del dominio social comprender cómo el comportamiento de los individuos afecta el comportamiento global del sistema. Así, podemos considerar todo este conocimiento como los *artefactos* que se producen en este paso.

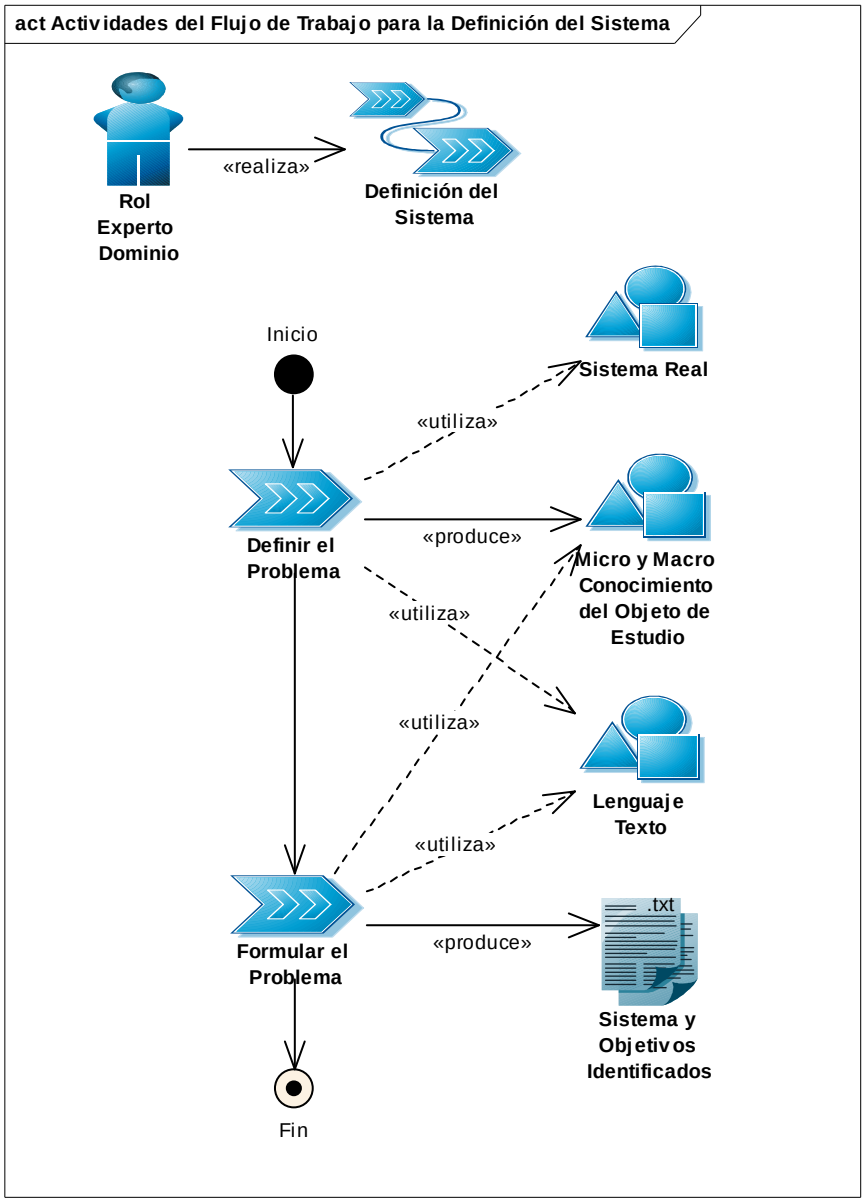

Figura 40. Actividades del Flujo de Trabajo para la Definición del Sistema

6.4.1.2. Fase de Ingeniería de SA

Esta fase consiste en transformar el conocimiento informal del experto en un programa ejecutable o modelo de ejecución. Para lograr esto, el sistema es especificado de forma abstracta a través de un modelo. Cualquier modelo útil exhibirá alguna forma de *abstracción* para permitir que los usuarios del modelo se concentren en los aspectos importantes del sistema, permitiéndoles así

poder gestionar la complejidad. Posteriormente, en base a esta abstracción se realiza una transformación de ese modelo a un modelo computacional ejecutable en las plataformas de simulación.

Flujo de Trabajo: Generación del Modelo de Diseño

Roles: Experto del Dominio

Entradas: Sistema y Objetivos Identificados

Salidas: Modelo de Diseño

Lenguajes y Herramientas: LMSA, IDK

Actividades:

1. *Abstracción*. La abstracción que ha de realizar el experto del dominio es proporcionada por medio de un modelo. Este modelo es principalmente una representación de los aspectos esenciales del sistema y por lo tanto contiene menos complejidad. Con esto, permite el análisis de propiedades específicas, la predicción de características del sistema, y facilita la comunicación con los roles involucrados en el proceso de desarrollo. Este trabajo constituye la información necesaria con la que se elaborará el *modelo de diseño*. Éste se realiza con el *micro-conocimiento* que tiene el experto de los agentes reales, su comportamiento y sus relaciones. Esta actividad es indicativa, principalmente se refiere a la separación de lo esencial de información adicional. En términos del desarrollo del software, lo esencial generalmente se refiere a la funcionalidad que ha de implementarse y la información adicional a aspectos tales como la plataforma en la cual eventualmente el software será desplegado. Sin embargo, la información adicional no dejan de ser importante. Ésta tiene que considerarse cuando se diseña y se desarrolla un sistema software, pero no requiere ser considerada por el experto del dominio, pues éste se encarga de cuestiones más fundamentales sobre el sistema. Esta información adicional se encuentra ya predefinida en los mecanismos y herramientas que proponemos para el desarrollo. Por lo tanto, al utilizar este marco metodológico se aceptan indirectamente los aspectos que considera esta información, por ejemplo, las plataformas de simulación seleccionadas. Una forma de abstracción es la selección de propiedades relevantes de las irrelevantes o aleatorias, lo cuál es conocido como *reducción* (Ludewig 2003). Otras formas importantes de abstracción son la generalización y la clasificación. La *generalización* se refiere a ignorar las diferencias entre elementos similares para formar una entidad en la cuál las similitudes puedan enfatizarse. La

clasificación[2] es el proceso de identificar tipos, también conocidos como conceptos (Peckham y Maryanski 1988).

2. *Definir Perspectivas del Modelo*. Esta actividad consiste en la *especificación* del *modelo de diseño* en el lenguaje LMSA. Esta especificación es una forma *preceptiva* de modelo. La tarea que ha de realizar el experto en este caso es la de traducir las abstracciones del sistema que ha realizado anteriormente utilizando conceptos de SMA. La idea es que el lenguaje LMSA orientado al modelado de agentes facilite esta tarea. Además, para facilitar el diseño del modelo se sugiere la creación de diferentes perspectivas del SMA. Para ello, el experto ha de familiarizarse con el lenguaje, y seguir la guía que se proporciona para la generación de estas perspectivas. En general, para concebir el SMA el experto debe pensar en los agentes como entidades con motivación, con objetivos. Los objetivos del sistema guían la estructuración del mismo en componentes que van a colaborar para satisfacerlos. De esta manera, se espera que el diseño sea más intuitivo, ya que esta forma de modelar y de razonar es más cercana al pensamiento humano. Para realizar el diseño de las perspectivas de una SA como SMA se llevan a cabo varias actividades. Éstas se realizan en *paralelo* y de forma *iterativa* e *incremental* hasta obtener el nivel de detalle deseado para cada una. Las perspectivas que se sugieren realizar para componer el *modelo de diseño* son las siguientes: la *perspectiva de organización*, las *perspectivas de agentes* necesarias, la *perspectiva de objetivos y tareas*, la *perspectiva de interacción*, la *perspectiva de roles y objetivos* y la *perspectiva de entorno*.

[2] La clasificación es la forma básica de abstracción encontrada en el modelado orientada a objetos, donde los tipos de objeto son los elementos principales de modelos conceptuales y las clases su respectiva realización en modelos de diseño.

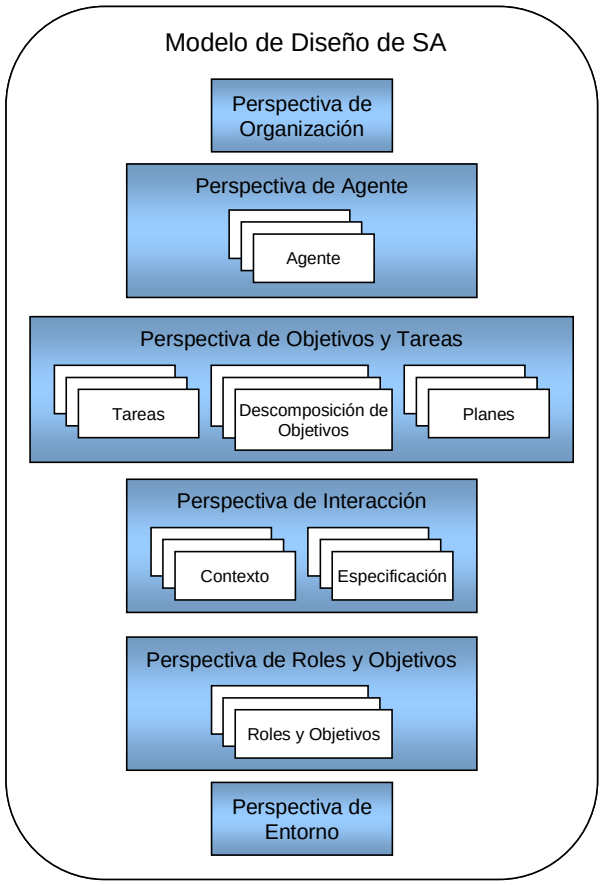

Figura 41. Perspectivas Producidas en la Especificación de un Modelo de Diseño de una SA

En la Figura 41 se muestran la relación de estas perspectivas. El detalle para la construcción de éstas se describe en (IDK 2004) y un ejemplo de su aplicación en el capítulo 7. Los pasos que componen esta actividad permiten generar la información necesaria para elaborar cada una de las perspectivas. Como se muestra en la Figura 42 estos pasos se realizan en paralelo, con constantes redefiniciones. Los pasos son:

2.1. Identificar Elementos de Organización

2.2. Identificar Propiedades de Agentes

2.3. Descomponer Objetivos y Tareas

2.4. Describir Tareas y Planes

2.5. Definir Parámetros de Adaptabilidad

2.6. Describir Interacciones

2.7. Describir Patrones de Estado Mental

2.8. Describir Entorno

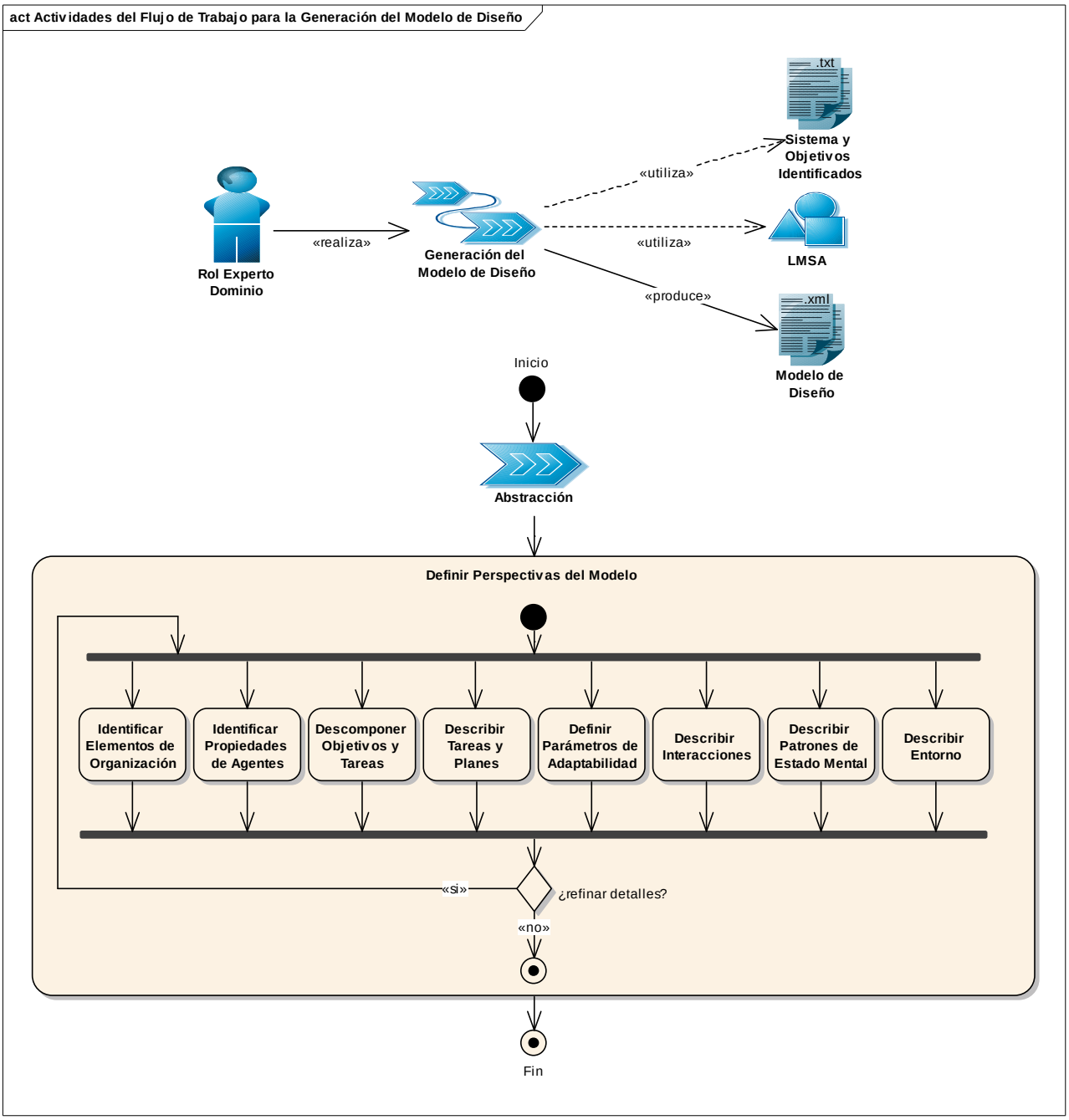

Figura 42. Actividades del Flujo de Trabajo para la Generación del Modelo de Diseño

Flujo de Trabajo: Generación del Modelo Específico de la Plataforma

Roles: Experto del Dominio, Ingeniero Software

Entradas: Modelo de Diseño

Salidas: Modelo de Ejecución

Lenguajes y Herramientas: Java, Script XML, Repast, Mason, IDK

Actividades:

1. *Desarrollo del módulo para la generación de código*. Aunque el módulo para la generación de código debe estar implementado antes de poder llevar a cabo este proceso metodológico, siempre puede ser ocurrir que el experto del dominio desee integrar más elementos de la especificación para su correspondiente representación en código automáticamente. Por lo tanto esta actividad puede implicar regenerar el módulo para la generación de código, con lo cual este proceso tendría que realizarse de nuevo (este proceso es descrito en la sección 5.4.1). Si no se requiere integrar nuevos elementos se realiza la siguiente actividad directamente.
2. *Generar Código Fuente*. La implementación de un modelo significa expresarlo en un nivel de abstracción muy bajo, es decir, en un nivel en el cual pueda comprenderse por un ordenador. Para esto, el *modelo de diseño* debe refinarse (reducir el nivel de abstracción) para obtener el producto de software final, el cual realiza el sistema deseado. Una vez que el modelo cumple con las propiedades requeridas se puede generar el código para una plataforma particular. Para ello, simplemente ha de invocar el módulo de generación de código correspondiente y obtener el *modelo de ejecución*.
3. *Compilación y Despliegue*. El código que se obtiene es fuente y habrá que compilarlo junto con las librerías (paquetes) de la plataforma de simulación. Una vez compilado puede desplegarse según sea la plataforma escogida. Para realizar estos dos pasos el experto del dominio puede usar los *scripts* definidos por el ingeniero software para automatizar estas tareas. A partir de este momento se puede utilizar el simulador y sus herramientas para ejecutar la simulación.

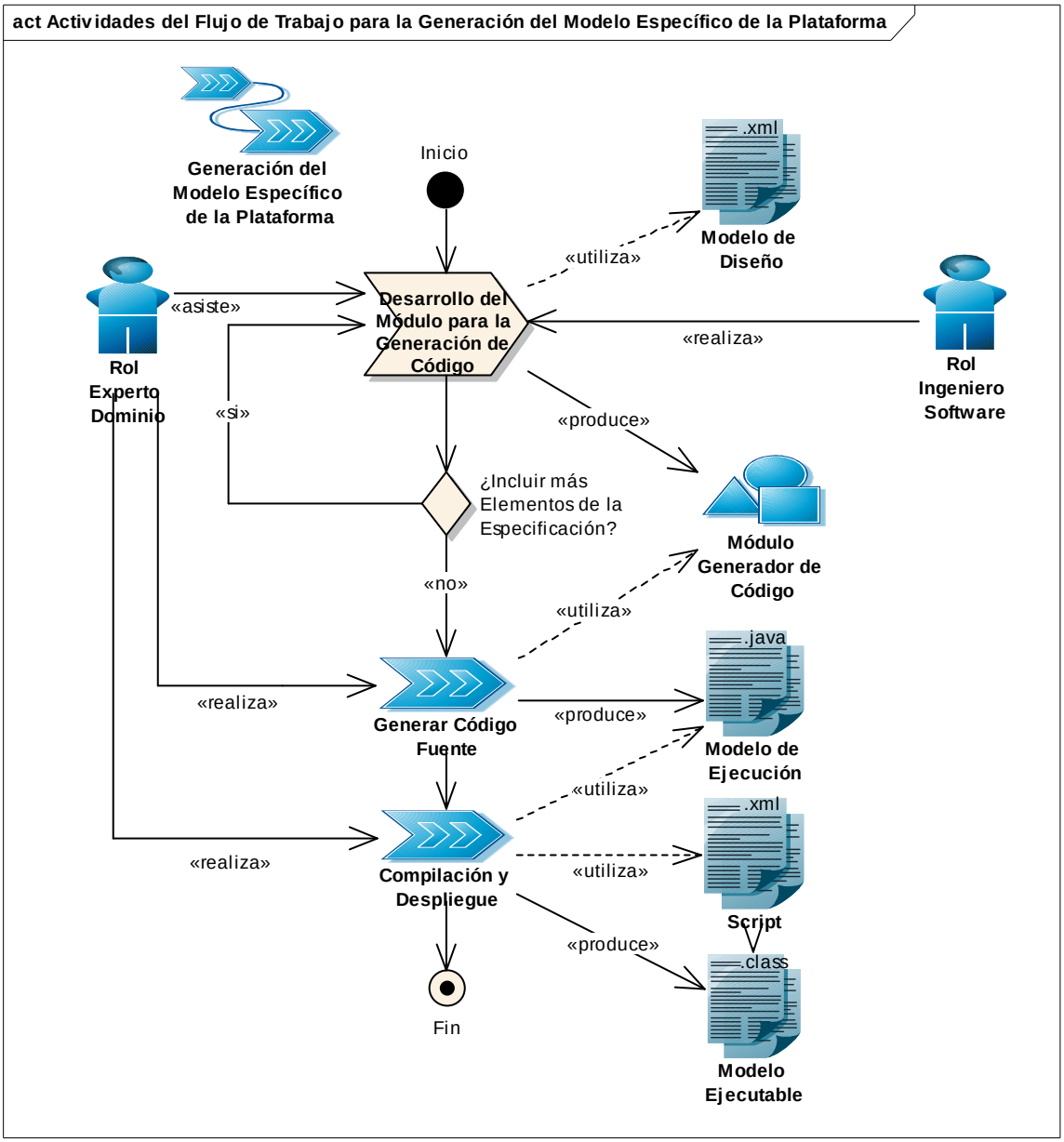

Figura 43. Actividades del Flujo de Trabajo para la Generación del Modelo Específico de la Plataforma

6.4.1.3. Fase de Experimentación

Esta fase consta de los pasos que complementan un estudio basado en la simulación. Una vez que se cuenta con el *modelo de ejecución* es posible ejecutarlo en la plataforma de simulación, de esta forma se puede observar el comportamiento del sistema simulado, y en base a los resultados que éste arroja es posible realizar los pasos complementarios de la investigación. En esta fase el *macro-conocimiento* del experto es fundamental, ya que permite llevar a cabo diferentes compro-

baciones, como la de la similitud estructural de los datos arrojados por el modelo de simulación con los del sistema real, o una comprobación de sensibilidad para analizar hasta qué punto el comportamiento de la simulación es sensible a las asunciones hechas durante el diseño.

Flujo de Trabajo: Estudio de Propiedades Emergentes

Roles: Experto del Dominio

Entradas: Modelo de Ejecución, Conocimiento Micro y Macro

Salidas: Modelo de Diseño

Lenguajes y Herramientas: Script XML, Repast, Mason, Módulo Generador de Documentación

Actividades:

1. *Validación Interna*. Antes de dar por válido un modelo de simulación y poder comparar los resultados simulados con los observados para comprobar las hipótesis del estudio, es necesario verificar que el comportamiento del modelo de ejecución o del programa corresponde con lo que el experto ha querido modelar en el diseño. Para llevar a cabo esta comprobación el experto debe realizar diferentes simulaciones de forma sistemática pues las simulaciones dependen de números seudo-aleatorios para simular los efectos de variables inmensurables y efectos aleatorios, y las simulaciones pueden producir resultados diferentes, no por esto erróneos. Con esto, el experto puede observar mejor el comportamiento del sistema y depurarlo si es necesario. Esta es una validación del modelo contra su implementación.
2. *Interpretación*. Una vez que el experto ha validado la implementación del modelo, el siguiente paso es el de la interpretación de los resultados. Qué el modelo sea lo que el experto ha querido diseñar no garantiza que el sistema simulado sea un buen modelo del sistema real, para esto se requiere la *validación externa*. En este paso el macro-conocimiento del experto es necesario para decidir si el comportamiento del modelo corresponde al comportamiento del fenómeno bajo estudio, para esto ha de formular distintos escenarios de simulación, llevarlos a cabo y comparar los datos arrojados por la simulación con los datos observados empíricamente y decidir si el modelo es válido. Aquí pueden realizarse múltiples validaciones. Por ejemplo, verificarse la *similitud estructural* (Gilbert y Troitzsch 1999), esto es, que haya una co-variación en circunstancias similares, lo cual puede contar como una evidencia de la adecuación del modelo. Una vez que se considera válido el modelo, el experto puede proseguir con la *interpretación* de los resultados, para esto lleva a cabo simulaciones bajo ciertas condiciones según las preguntas que ha formulado, si son de predicción seguramente ejecutará la simulación para simular su comportamiento en el futuro, mientras que si son especulativas se centrará en variar los valores de los parámetros o si

son teóricas observara cual de las asunciones que ha considerado en el modelo explican el fenómeno. En este último caso, se verifica lo que en la terminología de (Galán y Izquierdo 2005; Galán 2007) se conoce como *artefactos*. Esto es, asegurarse que no haya *desajustes* entre las hipótesis del modelo que realmente causan un fenómeno y las hipótesis que el experto cree son la causa de ese fenómeno. El marco metodológico facilita la interpretación de los resultados al experto del dominio ya que los elementos del modelo se han correspondido adecuadamente con elementos del código del modelo en el simulador.

3. *Replicación*. Una posibilidad que proporciona el marco metodológico es utilizar la misma especificación para generar código para distintas plataformas de simulación para replicar los resultados. Esto permite *alinear los modelos*, es decir, verificar que los modelos producen resultados equivalentes bajo condiciones equivalentes. Este paso es recomendable llevarlo a cabo con el modelo que se acaba de diseñar con el propósito de descubrir posibles sesgos causados por propiedades de las plataformas de simulación y que no son fáciles de observar normalmente, como se puede observarse en (Sansores y Pavón 2006). Con esto se puede mejorar la robustez de los resultados que van a publicarse. Esta actividad es asistida por el marco metodológico en el sentido de replicar el modelo en distintas plataformas para comparar resultados, sin embargo un proceso de replicación más riguroso (Galán et al. 2003) incluiría la replicación por investigadores independientes. Sin embargo, la replicación de resultados en diferentes plataformas ha resultado bastante ilustrativa para mejorar la confiabilidad de los modelos.
4. *Publicación de Resultados*. Finalmente la publicación de resultados consiste en colocar en algún formato adecuado la interpretación que se haya hecho del modelo de simulación ya sea por medio de las facilidades que presentan las plataformas o manualmente. Esto puede consistir de datos numéricos, descripciones textuales, gráficas, etc. Por otro lado, también se puede incluir en la información que se va a publicar la documentación del modelo de diseño con los diagramas de la especificación. Esto facilita la comunicación de los modelos pues la representación visual es más intuitiva para comprender que el código fuente, además facilita la replicación a terceros.

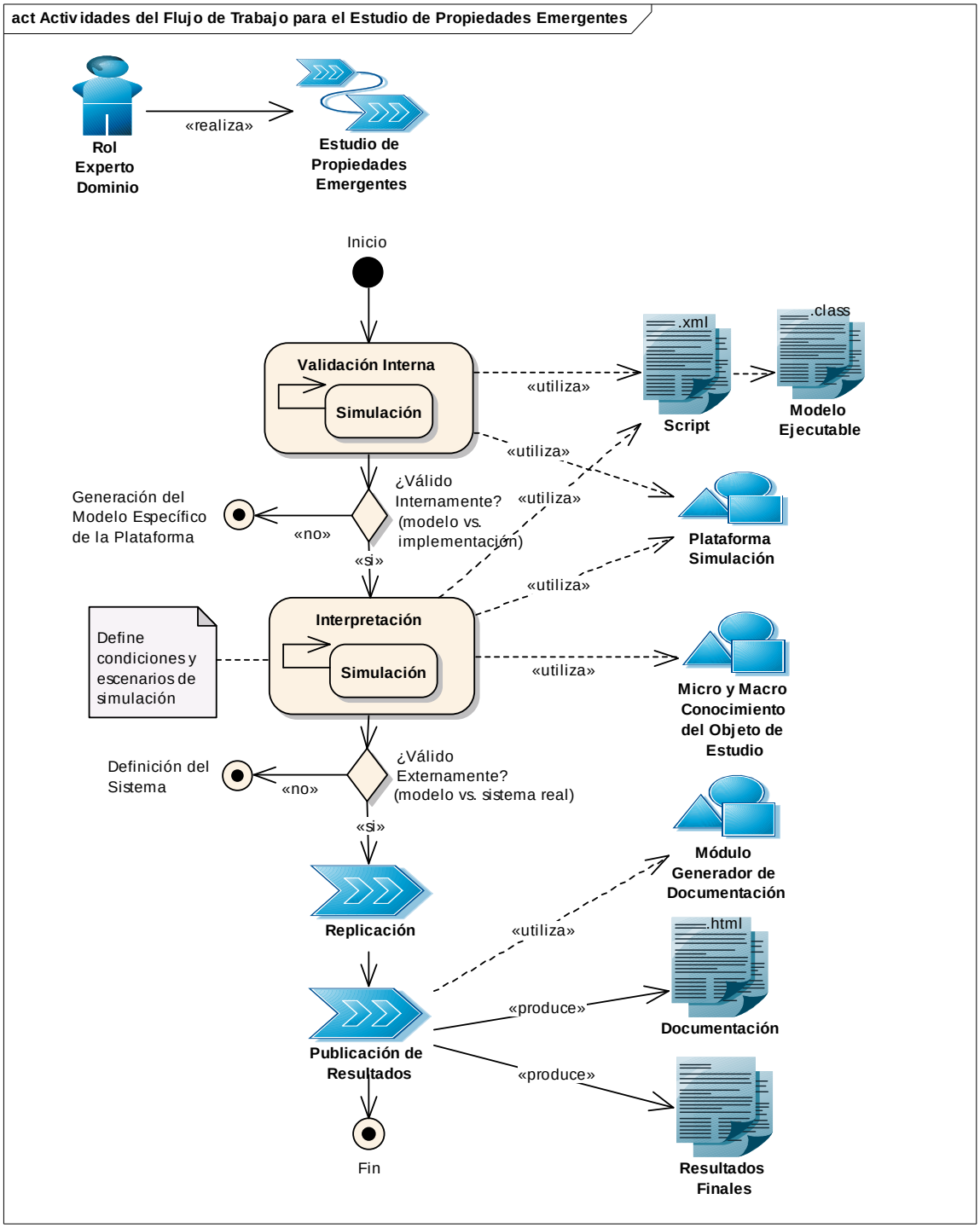

Figura 44. Actividades del Flujo de Trabajo para el Estudio de Propiedades Emergentes.

6.5. Conclusiones

En este capítulo se ha presentado el marco metodológico para el estudio de SA. Este marco es la propuesta de esta tesis para abordar la problemática encontrada en las metodologías y herramientas para la SSBA.

El marco propone una metodología y herramientas para facilitar su aplicación. La metodología establece un proceso de desarrollo basado en modelos y su integración a un proceso de investigación.

La metodología especifica cada actividad que ha de llevarse a cabo en ambos procesos, define los roles que participan en las actividades, establece cuáles son los artefactos que deben producirse en cada una de éstas y con qué lenguajes. Además, indica qué mecanismos y herramientas son necesarios en cada actividad.

Los beneficios que proporciona la utilización de esta metodología para el estudio de SA son:

1. Aborda todo el ciclo de vida de la investigación. La definición de roles en este ciclo permite que cada uno se centre en las actividades que mejor realiza. Además delimita las responsabilidades y mejora la comunicación entre los roles. Con esto, los expertos del dominio pueden enfocar su esfuerzo en el modelado y evaluación de resultados.
2. La colaboración que realizan los roles durante el proceso metodológico permite integrar el conocimiento de métodos, conceptos y prácticas de diferentes dominios como la simulación por ordenador, el paradigma de SMA, las ciencias sociales y la ingeniería del software conjuntamente para realizar un estudio de SA más fácilmente.
3. Existe una consistencia en todas las fases del estudio en términos de los conceptos de agentes y SMA utilizados.
4. Extensible. Se pueden integrar nuevos elementos en los dos niveles de abstracción propuestos para el desarrollo utilizando el mismo proceso metodológico, por ejemplo nuevas plataformas de simulación o nuevos elementos del lenguaje. Esto gracias a su orientación a la ingeniería MDE.

En general, el marco metodológico facilita el estudio de SA promoviendo primeramente la especificación de los modelos con un lenguaje de modelado, segundo, la realización de dicha especificación proporcionado las herramientas para facilitar su traducción a código de simulación y finalmente integrando este desarrollo a las actividades de experimentación para analizar patrones y procesos emergentes.

El marco metodológico tiene ciertas limitaciones relacionas con la complejidad del diseño, aunque éstas pueden reducirse. Por ejemplo, para alcanzar una automatización completa las perspectivas del modelo pueden llegar a ser bastante complejas. Una forma de atenuar este problema es lograr un compromiso entre la claridad de los modelos (p.ej. para que puedan comprenderse y

replicarse por terceros) y el nivel de detalle de la implementación que se ha de especificar en el modelo de diseño. Existen aspectos que pueden quererse incluir en la especificación del modelo como parte del lenguaje abstracto con tal de permitirle al experto del dominio definirlos, por ejemplo aspectos específicos de la plataforma. De esta forma, el experto podría controlar completamente la implementación del modelo. Sin embargo, esto causa que el diseño se complique por estas cuestiones irrelevantes que bien pueden afinarse posteriormente.

Es importante no poner en tensión la claridad de los modelos y el nivel de automatización considerable para la generación de código. Para ello, es importante recordar que el objetivo principal del marco metodológico es generar especificaciones claras que conduzcan más fácilmente a la implementación de los modelos.

Capítulo 7.

Experimentación

El propósito de este capítulo es ilustrar la aplicación del marco metodológico para el estudio de SA mediante la elaboración de un caso de estudio. Éste consiste en un estudio de simulación sobre el altruismo. El ejemplo es interesante ya que muestra la importancia del modelado de agentes como entidades cognitivas y remarca el impacto de la inteligencia y de los sistemas basados en objetivos en la propagación del altruismo. El sistema está inspirado en un famoso estudio etológico sobre el altruismo entre murciélagos (Wilkinson 1984). La experimentación de este capítulo incluye la aplicación de la metodología para el estudio de este sistema, su especificación con el lenguaje LMSA, el proceso de traducción de los modelos a código de simulación, y una evaluación de los resultados en base a la replicación del modelo en distintas plataformas. El capítulo concluye con una discusión de los beneficios y limitaciones de la propuesta como resultado de su aplicación integral en el caso de estudio.

7.1. Introducción

Este capítulo presenta la experimentación realizada para evaluar las propuestas de esta tesis. Se trata de un caso de estudio que desarrolla la especificación, implementación y análisis de una SA

de agentes inteligentes. El propósito de este caso de estudio es evaluar las ventajas y limitaciones del uso del marco metodológico.

Anteriormente se discutió que existen casos en la simulación social en los que son necesarios modelos de agentes cognitivos (véase el capítulo 2 y (Sun 2005)), sobre todo cuando el análisis de este micro nivel restringe de alguna forma los fenómenos que surgen en el macro nivel. Para realizar el análisis a nivel cognitivo el marco metodológico ha adoptado un modelo de SMA de agentes inteligentes que se especifica con el lenguaje LMSA. Para probar la viabilidad de este lenguaje y su capacidad expresiva se ha seleccionado un caso de estudio que requiere considerar agentes racionales e intencionales con la finalidad de analizar cómo ciertas estructuras mentales como objetivos, motivaciones, etc. influyen en el comportamiento altruista de estos últimos.

Para evaluar los mecanismos de implementación del modelo de simulación se realizará la generación de código para las dos plataformas de simulación adoptadas por el marco. La replicación en ambas plataformas permitirá comparar resultados y estudiar el efecto que pueden tener las facilidades que proporciona cada herramienta sobre éstos, por ejemplo, el efecto que tienen las diferentes estrategias de planificación de las plataformas. En este mismo caso de estudio también es exigida la capacidad de adaptación de los agentes. Con esto será posible evaluar la facilidad de aplicación del mecanismo de adaptabilidad que proporciona el marco. La fidelidad del modelo computacional SMA con respecto a su correspondiente modelo conceptual también se pone en evidencia durante la generación de código, con ello se podrá comprobar su suficiencia o si es necesario ampliarlo.

Finalmente, la metodología será evaluada durante la realización de las actividades que ésta plantea para el estudio de SA, entre éstas se incluyen las de desarrollo mencionadas anteriormente y las de análisis del modelo.

El caso de estudio que se ha elegido se basa en el modelo de simulación presentado originalmente en (Di Tosto et al. 2006) y que se replica en este trabajo. El modelo consiste en un ejemplo inspirado en un estudio etológico sobre murciélagos en América Central y sus hábitos para compartir comida. En este ejemplo se comparan diferentes arquitecturas de agentes (con diferentes niveles de complejidad cognitiva) en la evolución del hábito para compartir comida y se analiza su desempeño en la presencia de tramposos en la población. El experimento está orientado a estudiar el rol de la modelos de agentes cognitivos en el altruismo. La replicación de este modelo ha representado un reto ya que la versión original no cuenta con una especificación abstracta y completa, el modelo original solo describe el problema de forma textual y hay algunas lagunas de información que fue necesario completar por medio de la experimentación. Para el caso que nos interesa destacar en este trabajo, sólo se presenta la replicación del modelo que considera agentes cognitivos.

El capítulo incluye las siguientes secciones. La sección 7.2 presenta la fase de identificación del objeto de estudio. Ésta consiste de la descripción del caso de estudio tal y cómo se plantea en (Di

Tosto et al. 2006), incluyendo los objetivos de su aplicación. La sección 7.3 presenta la fase de ingeniería de SA, que comprende la especificación del modelo SMA del caso de estudio y la implementación de la simulación. La sección 7.4 presenta la fase de experimentación, analiza los resultados obtenidos comparando los resultados originales y los replicados en las dos plataformas de simulación. Finalmente, la sección 7.5 presenta una valoración de la metodología en vista de su aplicación.

7.2. Fase de Identificación del Objeto de Estudio

En (Di Tosto et al. 2006) se argumenta que el altruismo se ha estudiado desde diferentes disciplinas y perspectivas, y que un acuerdo común en estos estudios es que el altruismo es ventajoso desde un punto de vista biológico. Aunque una forma pura e incondicional de altruismo no tiene oportunidad de extenderse en la presencia de tramposos. Por el contrario, se concluye que lo que contribuye al éxito reproductivo de los individuos (y grupos) es una forma condicionada de altruismo, es decir, la reciprocidad o actuar para el beneficio de otros, conocidos como altruistas.

Con respecto a la reciprocidad, en el estudio de Di Tosto et al. se cuestionan las siguientes preguntas: ¿Cómo seleccionan los altruistas quiénes recibirán su ayuda? ¿Cómo poder decir si los actuales receptores de su ayuda serán futuros donadores? Esencialmente, ya sea la experiencia directa (la imagen que se tenga del receptor potencial) o la indirecta (su reputación) ayudan a seleccionar a los receptores. Además, en este estudio la norma de reciprocidad es vista como un mecanismo *mental* usado para el razonamiento social y la acción social. Por lo tanto, se cuestionan igualmente: ¿Qué tipo de construcción mental corresponde a la norma de reciprocidad (y posiblemente, a cualquier otra norma social)?

Según (Di Tosto et al. 2006) esta pregunta ha recibido poca atención en el estudio del altruismo basado en la simulación. Según estos autores, existe un consistente número de trabajos en el campo que utilizan el altruismo para afrontar problemas de optimización en SMA o lo utilizan en la comunicación indirecta basada en la estigmergia para observar la emergencia de comportamientos de cooperación en especies similares a las hormigas. El enfoque de (Di Tosto et al. 2006) está dirigido al interior de la mente de los agentes, y a esclarecer el rol que juegan los sistemas inteligentes en la evolución del comportamiento pro-social. Más que proporcionar una respuesta exhaustiva, el estudio se centra en la comparación entre dos opciones extremas: un algoritmo altruista simple o sub-cognitivo, y un algoritmo altruista inteligente o casi-cognitivo. En la primera, los agentes aplican rutinas bajo condiciones dadas. En la segunda, los agentes ejecutan acciones para lograr sus objetivos. De esta forma, se intenta modelar motivaciones y observar su impacto sobre el altruismo.

Según Di Tosto et al., la escasa atención prestada a la mente de los agentes en el estudio del altruismo basado en la simulación puede deberse a un teorema popular, según el cual el altruismo y la inteligencia son esencialmente incompatibles. La observación común de que el altruismo es frecuente en especies similares a las hormigas, donde la autonomía individual no está completamente desarrollada y el comportamiento está programado rígidamente por la evolución biológica, lleva a la suposición general de que mientras mayor sea el rango evolutivo de la especie, menos frecuente será el comportamiento altruista. Así, los autores se hacen las siguientes preguntas: ¿Quiere decir que la inteligencia dificulta o inclusive previene el altruismo? ¿Cuáles son los efectos de la complejidad mental y de la autonomía individual sobre la acción social?

Para verificar la validez del teorema popular previo, los autores ejecutan una serie de experimentos de simulación inspirados en un famoso estudio etológico (Wilkinson 1984) sobre el altruismo de los murciélagos. Los murciélagos desarrollaron una forma única de compartir alimento, que consiste en que los cazadores exitosos regurgitan una porción de la sangre injerida a favor de sus compañeros menos afortunados. Los biólogos sociales han interpretado la compartición de alimento como evidencia para la teoría del altruismo recíproco. Estas especies ofrecen un buen objeto de estudio para modelar el altruismo ya que su supervivencia está fuertemente ligada a la evolución del comportamiento altruista. Así, la intención de Di Tosto et al. es explorar el altruismo a un nivel abstracto, y descubrir sus ingredientes mentales subyacentes. Lejos de dirigir la cuestión a cómo es en realidad la mente del murciélago, se explora la relación entre algún aspecto de cognición y altruismo, mostrando probablemente su co-evolución. En resumen, el estudio intenta:

- Modelar el altruismo a un nivel abstracto, inspirándose en un ejemplo del mundo real (murciélagos) que proporciona evidencia para establecer parámetros a valores dados en una forma no arbitraria.
- Explorar los ingredientes internos permitiendo a los agentes exhibir el altruismo; en particular comparar dos algoritmos diferentes, uno sencillo y otro inteligente, y observar sus resultados respectivos.
- Comprobar la validez del teorema popular que dice que la inteligencia obstaculiza el altruismo: ¿Debe asumirse qué los murciélagos son lo suficientemente estúpidos para aplicar rígidamente algunas rutinas pre-establecidas, o de lo contrario es su comportamiento compatible con un equipo mental más inteligente y sofisticado? Y si es el caso, ¿cuáles son las ventajas específicas de esa complejidad mental superior?

El modelo considera una población de agentes o murciélagos que viven en cuevas a donde regresan después de cazar y en donde realizan actividades sociales como compartir alimento y limpiarse. Las entidades explícitamente modeladas como agentes son los murciélagos. Cada ciclo de la simulación incluye una etapa de día y otra de noche. Durante el día, los agentes realizan activi-

dades sociales, en la noche cazan. En el modelo la caza se define como un parámetro ecológico de acuerdo a datos reales, el valor por defectos es de 93%. Así, cada noche el 93% de la población encontrará alimento para sobrevivir hasta la siguiente caza. El resto 7% de la población morirá de hambre, a menos que reciban ayuda de algún otro compañero (en forma de regurgitación).

Los murciélagos no acumulan recursos, la caza se realiza solamente para consumo inmediato. Cuando cazan obtienen una autonomía de no más de 60 horas. La actividad de limpieza que realizan los murciélagos en la naturaleza tiene al menos dos efectos, uno es familiarizarse y otro verificar su estado físico para saber si han cazado. Por lo tanto, en el modelo los agentes están inmersos en una red de limpieza que se activa de forma aleatoria al inicio de la simulación. Cada día se forman pares de agentes cuando cada agente escoge un compañero de su red. Así, la red de *limpieza* tiene el efecto de incrementar la probabilidad de que los murciélagos compartan alimento con otros de la misma cueva, un murciélago con hambre pedirá ayuda a su compañero de limpieza y evitará la muerte si alguno ha cazado, sólo una vez es posible pedir ayuda, si se le niega morirá. Además de la familiaridad, la red de limpieza crea una base de reciprocidad. El hecho de prestar ayuda permite a los agentes acumular créditos, que serán extinguidos cuando la ayuda les sea devuelta. La red facilita el reencuentro y por lo tanto la extinción de los créditos.

Los agentes son provistos con conocimiento social que consiste de una memoria de interacciones de limpieza, de compartición de alimento pasados y de su crédito consecuente. Después de cada simulación, el número de murciélagos muertos, el número de actos altruistas realizados y el número de créditos generados y devueltos es registrado.

En este capítulo se presenta la replicación de una parte de este estudio. La parte más interesante para este trabajo de experimentación es la exploración de la mente de los agentes, es decir, como influyen las estructuras mentales en su comportamiento social. Para ello, nos centramos en replicar el caso de los agentes inteligentes, y comparar los resultados de la replicación.

7.3. Fase de Ingeniería de SA

La especificación del modelo se realiza siguiendo el proceso establecido por el flujo de trabajo *generación del modelo de diseño*. Las perspectivas que componen la especificación se han desarrollado en paralelo y el orden de presentación de éstas no guarda ninguna relación con el orden de su creación (el proceso es iterativo y las distintas vistas del modelo se retroalimentan).

7.3.1. Modelado

La Figura 45 ilustra la perspectiva organización. En ésta se estructura el modelo de la SA y se define el contexto en el que conviven los agentes y en el cual desempeñan sus roles. El *Modelo* de

simulación está representado por una entidad *organización*. El modelo tiene dos *grupos*, *Cueva* y un grupo especial que es una extensión del lenguaje LMSA que representa una *Red Social*. Los miembros del grupo *Cueva* tienen la oportunidad de desempeñar dos roles, uno *Altruista* y otro *Recipiente* o que recibe ayuda. Existe un solo tipo de agente en el modelo, este es el *Murciélago*. Este agente pertenece a los dos grupos que se definen en el modelo. El hecho de pertenecer al grupo *Red Social* le da la oportunidad de contar con la funcionalidad predefinida para los miembros de este grupo, como el poder formar una red de interacciones y una memoria de éstas. Cada nodo en la red representará para este caso particular un recipiente de ayuda y los enlaces una acción de compartición de alimentos.

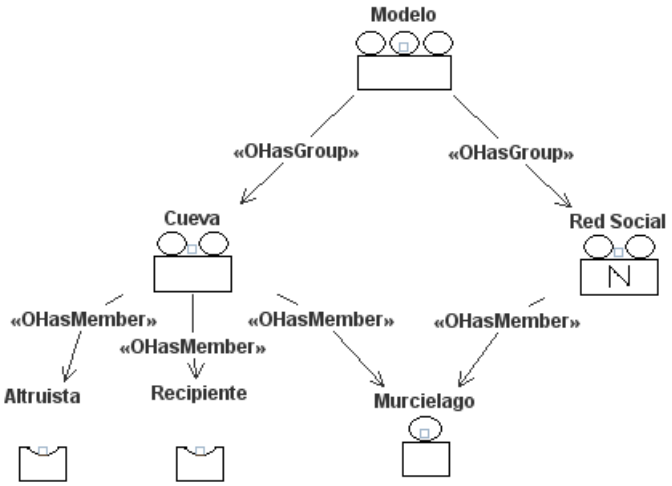

Figura 45. Perspectiva Organización del Modelo Altruismo con Agentes Inteligentes

La siguiente perspectiva describe cada tipo de agente en el modelo. La Figura 46 ilustra la perspectiva agente del tipo *Murciélago*. Esta perspectiva agente considera para su composición que un agente es una entidad autónoma que se guía por el principio de racionalidad. Para ello, toma en cuenta la funcionalidad de cada agente incluyendo la siguiente información: los *objetivos* que el agente persigue, las *tareas* que se supone debe realizar, los *roles* que va a desempeñar. El comportamiento es definido por tres componentes: *gestor de estado mental*, *procesador de estado mental* y el *estado mental*, como agregación de entidades mentales como objetivos, creencias y hechos. Cada agente tiene un estado mental inicial, representado por una asociación del agente a una entidad de estado mental.

A nivel de la especificación del SMA, el procesador y el gestor están definidos con un alto nivel de abstracción. Es el proceso de generación de código el que va a definir la realización concreta de ambos sobre la plataforma final. Para ambas entidades se ha realizado una implementación concre-

ta en la librería que se añade a las plataformas de simulación. En la Figura 46 se muestra la representación gráfica de los elementos que son utilizados para describir el tipo de agente murciélago.

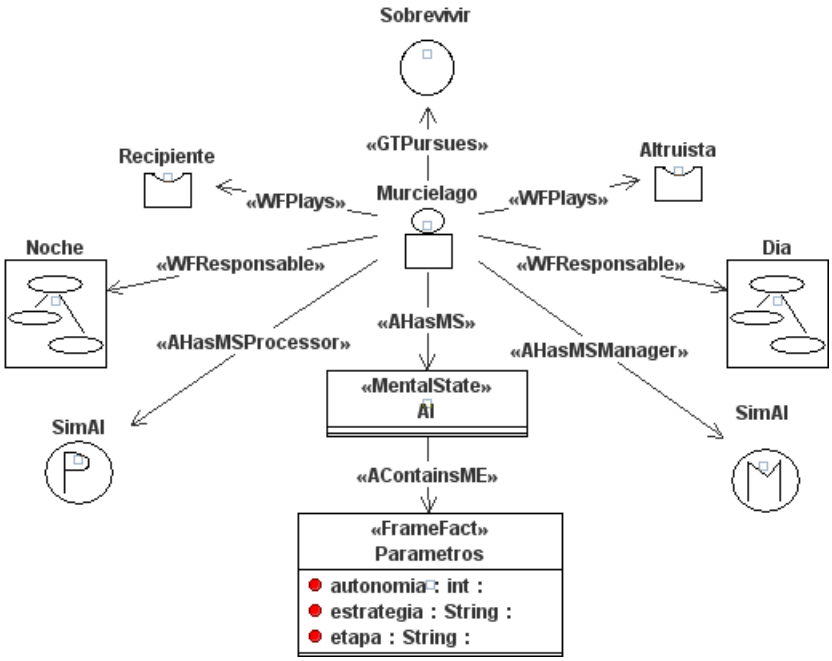

Figura 46. Perspectiva Agente del Modelo Altruismo con Agentes Inteligentes

El comportamiento de los murciélagos es representado colectivamente como planes, el plan *Dia* y plan *Noche*. Estos planes después son descompuestos en tareas más simples. El agente persigue el objetivo *Sobrevivir* y puede desempeña dos roles, *altruista* y *recipiente*. El estado mental del agente muestra algunos hechos con los que cuenta inicialmente, el procesador y gestor del estado mental (*SimAI*) se encargan de producir las decisiones del agente. El gestor proporciona operaciones para crear, borrar y modificar entidades mentales y el procesador toma decisiones basado en objetivos y tareas que los satisfacen.

La siguiente perspectiva es la de objetivos y tareas. Ésta se descompone en la de *tareas*, en el de descomposición de *objetivos* y su relación con las *tareas*, y adicionalmente en la de *planes*, que es la secuencia de tareas que realiza un agente. Esta última perspectiva se ha añadido para soportar el diseño que se propone en el marco metodológico para la SSBA.

La perspectiva objetivos y tareas considera la descomposición de objetivos con miras a disminuir su complejidad, y describe las consecuencias de ejecutar una tarea y porque ésta debe ser realizada, p.ej. justifica la ejecución de tareas como un medio de satisfacer los objetivos. Las relaciones de satisfacción y fracaso permiten identificar que objetivos están influenciados por la ejecución de una tarea. En esta perspectiva se explica como la resolución de un objetivo puede

afectar la resolución de otros objetivos por relaciones de descomposición y dependencia. La Figura 47 ilustra esta perspectiva y sus elementos.

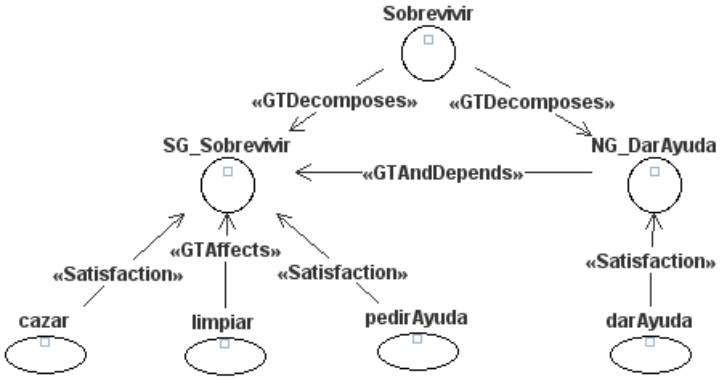

Figura 47. Perspectiva Objetivos y Tareas: Descomposición de Objetivos y Tareas del Modelo Altruismo con Agentes Inteligentes

La descomposición de objetivos y tareas es guiada por el *principio de racionalidad* en el cual las acciones de los agentes son justificadas por los objetivos que persigue. La asociación consiste en instancias de relaciones como *Satisface* o *Afecta*. Esta asociación constituye la justificación de la ejecución de la tarea. En estos dos casos es necesario además indicar bajo qué condiciones se asume el éxito o fracaso del objetivo. Estas condiciones se expresan con *patrones de estado mental*. Esta descomposición se acompaña además de dependencias entre objetivos.

En el caso particular del modelo de simulación que se especifica, el agente *Murciélago* persigue el objetivo *Sobrevivir*, que se descompone en dos objetivos, uno que es el de prestar ayuda, *NG_DarAyuda,* y otro, *SG_Sobrevivir*, mantenerse con vida. Ambos objetivos dependen uno del otro. Para el objetivo *SG_Sobrevivir* se asocian varias tareas, como la de *cazar* que satisface este objetivo, *limpiar* que afecta el objetivo, y *pedirAyuda* que también puede satisfacerlo. Para ambos casos de satisfacción se ha de definir un patrón de estado mental que indique la condición de satisfacción. Para satisfacer el objetivo *NG_DarAyuda* se asocia la tarea *darAyuda*.

El *patrón de estado mental*, dentro de la perspectiva agente, sustituye la entidad o el tipo *agente* por el de *agente concreto* para describir el comportamiento del agente en un instante concreto. La Figura 48 muestra el patrón de estado mental necesario para satisfacer el objetivo *SG_Sobrevivir* mediante la tarea *cazar*. En la figura puede observarse un agente concreto *Murciélago*, con un estado mental, el cual requiere contar con un hecho específico. Esta descripción indica la condición que se ha de cumplir para alcanzar el objetivo *SG_Sobrevivir*, en este caso, contar con el hecho *autonomía* y que éste tenga el valor mayor de 60.

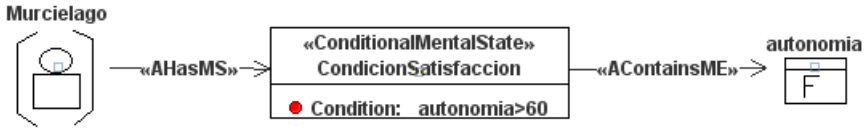

Figura 48. Patrón de Estado Mental para Satisfacer el Objetivo SG_Sobrevivir en el Modelo Altruismo con Agentes Inteligentes

La perspectiva planes se ilustra en la Figura 49. Ésta describe simplemente que el plan *Noche* se descompone en una tarea más simple *cazar* y el plan *Día* se descompone en otras como *limpiar*, *pedirAyuda*, etc. Ambos planes están asociados a una entidad mental del agente que indica si la *etapa* es de día o de noche, lo que determina qué plan ejecutar en un momento dado.

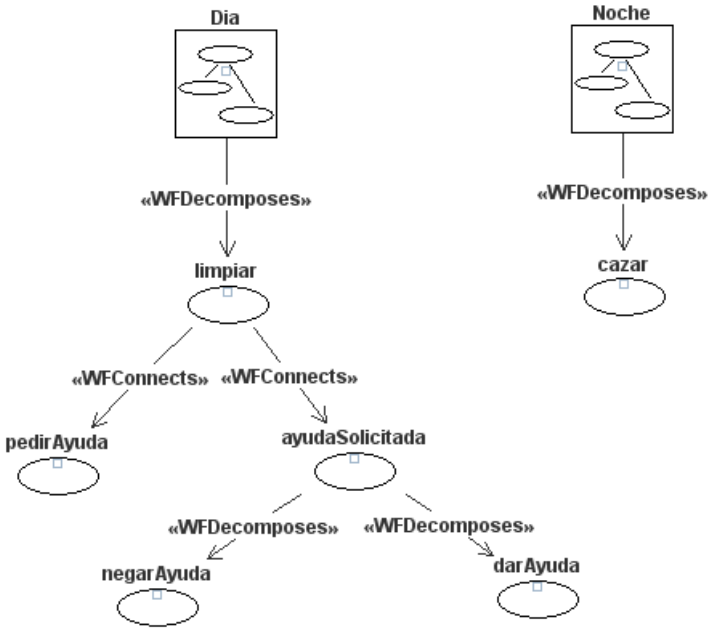

Fig. 49. Perspectiva Objetivos y Tareas: Planes del Modelo Altruismo con Agentes Inteligentes

Ambos planes o secuencias de tareas se ilustran en la Figura 50 donde se describe el detalle de sus precondiciones y post-condiciones y las consecuencias de su realización. Las tareas pueden iniciar cuando ciertas precondiciones, expresadas en términos de entidades del estado mental del agente son satisfechas. Así, es posible decorar cada relación con un *patrón de estado mental,* para indicar en qué condiciones puede tener lugar la realización de la tarea. Cada tarea es finalmente asociada a un agente, que es el responsable de la puesta en marcha, del control y de la ejecución de la tarea.

Una vez que el plan *Noche* entra en ejecución (cuando el estado mental del agente cuenta con el *hecho etapa==noche*) el agente puede realizar las tareas del plan. En la Figura 50 se muestra que

para la ejecución de la tarea *cazar* del plan *Noche* es necesario una entidad que representa una aplicación interna, es decir, el código que implementa esta funcionalidad (*CazarAleatoriamente-Code*) con las condiciones necesarias, p.ej. de forma aleatoria con los parámetros preestablecidos. Esta tarea produce o afecta dos entidades mentales, la *autonomía* y el *rol* que desempeña el agente. El estado mental del agente cuando caza y afecta el *hecho* rol que desempeña el agente en un momento dado se ilustra en la Figura 51. En esta figura puede observarse el estado mental condicionado del agente: si la *autonomía* tiene un valor menor o igual a 12 entonces desempeña el rol *Recipiente*, de otra forma el rol *Altruista*.

En esta misma perspectiva tareas se ilustran las tareas que componen el plan *Día* (que entra en ejecución cuando el estado mental del agente cuenta con el *hecho etapa==día*). La primera tarea de este plan es la de *limpiar*. También se muestran las precondiciones de esta tarea necesarias para su ejecución, por ejemplo la entidad aplicación interna con el código que implementa la funcionalidad para que un agente pueda interactuar en la red social (*BuscaCompañeroRedSocialCode*) (p.ej. para buscar un compañero). La tarea *limpiar* conecta con la tarea *pedirAyuda* siempre y cuando la condición de que el agente esté desempeñando el rol *Recipiente* se cumpla. Esto se muestra con la entidad mental *rol* como precondición de la tarea *pedirAyuda* y la condición del estado mental sobre esta relación llamada "recipiente". Así, existe un patrón de estado mental que describe esta condición. Finalmente, esta misma tarea produce una interacción llamada *pedirAyudaAltruista* que se describe en la perspectiva de interacción. La tarea *ayudaSolicitada* es ejecutada cuando el agente desempeña el rol *altruista* y algún otro agente le pide ayuda. Esta tarea conecta con otras dos tareas del mismo plan *día* dependiendo de la estrategia que desempeñe el agente. De esta manera, puede ejecutarse la tarea *darAyuda* y *negarAyuda*. Las estrategias que desempeña el agente dependen de la motivación con la que persigue sus objetivos y se describen en la perspectiva roles y objetivos.

La perspectiva roles y objetivos se ilustra en la Figura 52. Esta perspectiva se ha añadido como parte del modelo de diseño del proceso metodológico propuesto para la SSBA. El propósito de este modelo es el de especificar ciertos aspectos que componen el mecanismo de auto-organización basado en la capacidad de los agentes de modificar dinámicamente su comportamiento por medio de un *refuerzo* (el funcionamiento completo de este mecanismo de adaptabilidad es explicado en la sección 5.3) que se aplica a la fuerza motivacional para perseguir un objetivo. Los elementos básicos para especificar este mecanismo son: la definición de las interacciones que significan un refuerzo positivo que aumentan el comportamiento del agente y las que significan un refuerzo negativo que lo disminuyen. Esta especificación se realiza en la perspectiva interacción. De esta forma los agentes se adaptan y se observa la especialización de roles al seleccionar dinámicamente un nuevo comportamiento dependiendo del valor de su fuerza motivacional para perseguir objetivos.

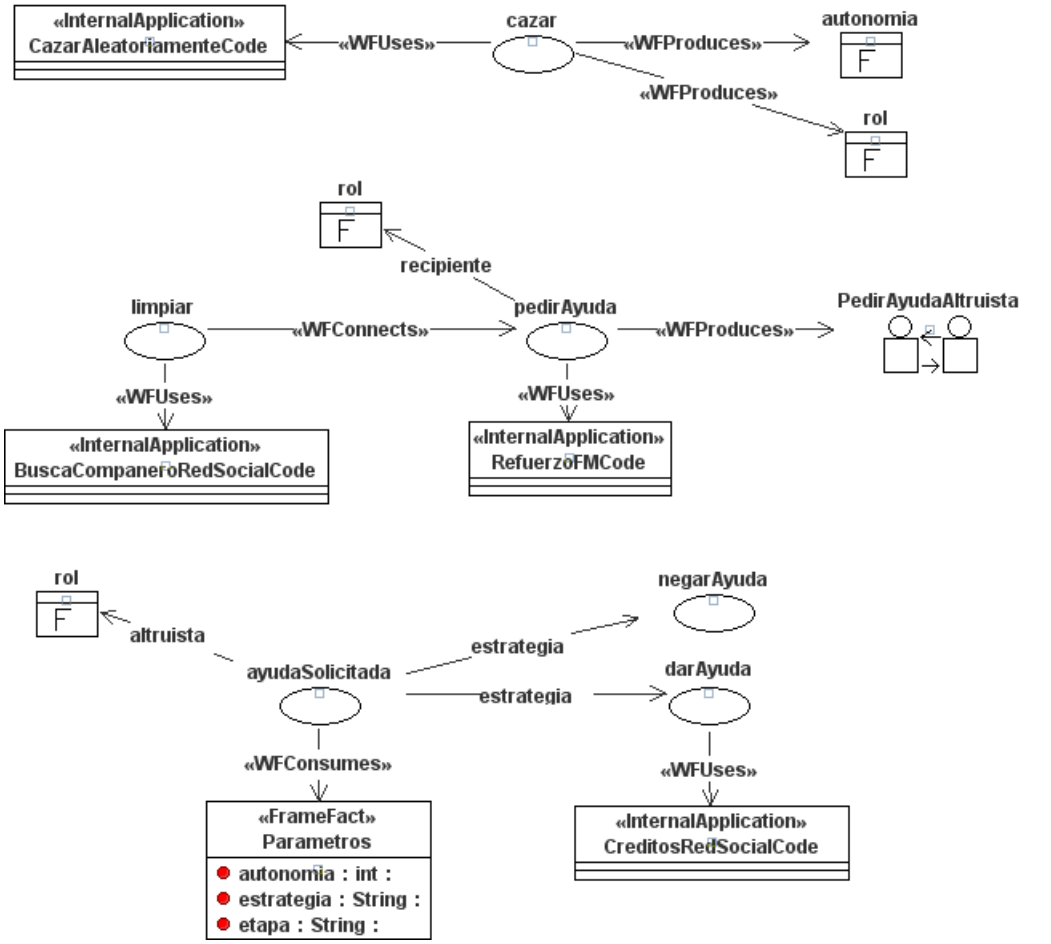

Figura 50. Perspectiva Objetivos y Tareas: Tareas del Modelo Altruismo con Agentes Inteligentes

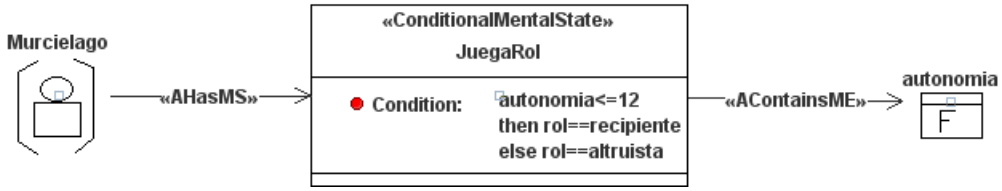

Figura 51. Patrón de Estado Mental para Establecer el Rol que Desempeña el Agente en el Modelo Altruismo con Agentes Inteligentes

Para poder especificar la asociación de la fuerza motivacional con el objetivo que persigue un agente y la especialización de roles que puede jugar un agente dependiendo de esta fuerza motivacional se crea la perspectiva roles y objetivos que se muestra en la Figura 52. Esta perspectiva se compone del rol principal y de la especialización de roles por medio de relaciones de *herencia*.

También puede observarse el objetivo y la motivación para alcanzarlo, dependiendo de esta motivación se define la condición para adoptar el rol o comportamiento que tendrá el agente.

Específicamente, en la versión del modelo de (Di Tosto et al. 2006) con agentes inteligentes, el agente persigue los objetivos *NG_DarAyuda* y *SG_Sobrevivir* y ambos cuentan con una fuerza motivacional que varía según es reforzado su comportamiento. Un agente que desempeña el rol *altruista* persigue el objetivo *NG_DarAyuda* y la ayuda que presta depende de la estrategia que desempeñe el agente. El objetivo *SG_Sobrevivir* no varía y se fija con el valor cero. Las estrategias con las que se relaciona la ayuda que presta el agente (es decir, si presta o no ayuda y la cantidad de alimento que dona) son: *tramposo*, *prudente*, *justo*, *generoso*, y *mártir*. Estas características se especifican en el diagrama de la Figura 52. En ésta se indica que un agente puede desempeñar el rol principal *Altruista* y otro rol *Recipiente*. Cuando desempeña el rol *altruista* este se puede especializar dependiendo del valor de la fuerza *fm* con la que es perseguido el objetivo correspondiente, en este caso el objetivo *NG_DarAyuda*. Así, si *fm<-2* el agente jugará la estrategia *tramposo*, si *-2<=fm<0* la estrategia será *prudente*, si *0<=fm<=1* la estrategia será *justo*, si 1<fm<=4 la estrategia será *generoso*, y si *4<fm* la estrategia será *mártir*. Con esto se refleja si el agente tiene menos o mayor motivación de alcanzar cierto objetivo. Si la motivación cambia el agente cambiará igualmente de comportamiento.

El valor de *fm* es incrementado por actos altruistas, es decir, cuando se aplica la norma del altruismo y más aún cuando esos actos son reciprocados. Mientras que el incremento de donaciones no-reciprocadas lo disminuyen.

Un agente que desempeñe la estrategia *tramposo* siempre negará ayuda, si es *prudente* donará 6 horas de autonomía si cuenta con 48 horas, si cuenta con 24 negará ayuda. Si el agente es *justo* donará 12 horas de su autonomía si cuenta con 48 y 6 si cuenta con 24. Si es *generoso* donará 24 horas de su autonomía si cuenta con 48 y 12 horas si cuenta con 24. Cuando es *mártir* donará aún cuando sólo cuente con 12 horas de autonomía.

La perspectiva interacción se ilustra en la Figura 53. Esta perspectiva simplemente muestra el intercambio de mensajes que realizan los roles *recipiente* y *altruista* durante la interacción que se produjo cuando el *recipiente* pide ayuda y las tareas que se realizan durante la interacción. La realización de las tareas depende del estado mental del agente, es decir, si presta ayuda o la niega. Las aplicaciones internas (*CreditosRedSocialCode* y *RefuerzoFMCode*) que se ilustran en la Figura 50 se utilizan durante la ejecución de estas tareas para actualizar la red social y el valor de *fm* según hayan habido actos altruistas o no.

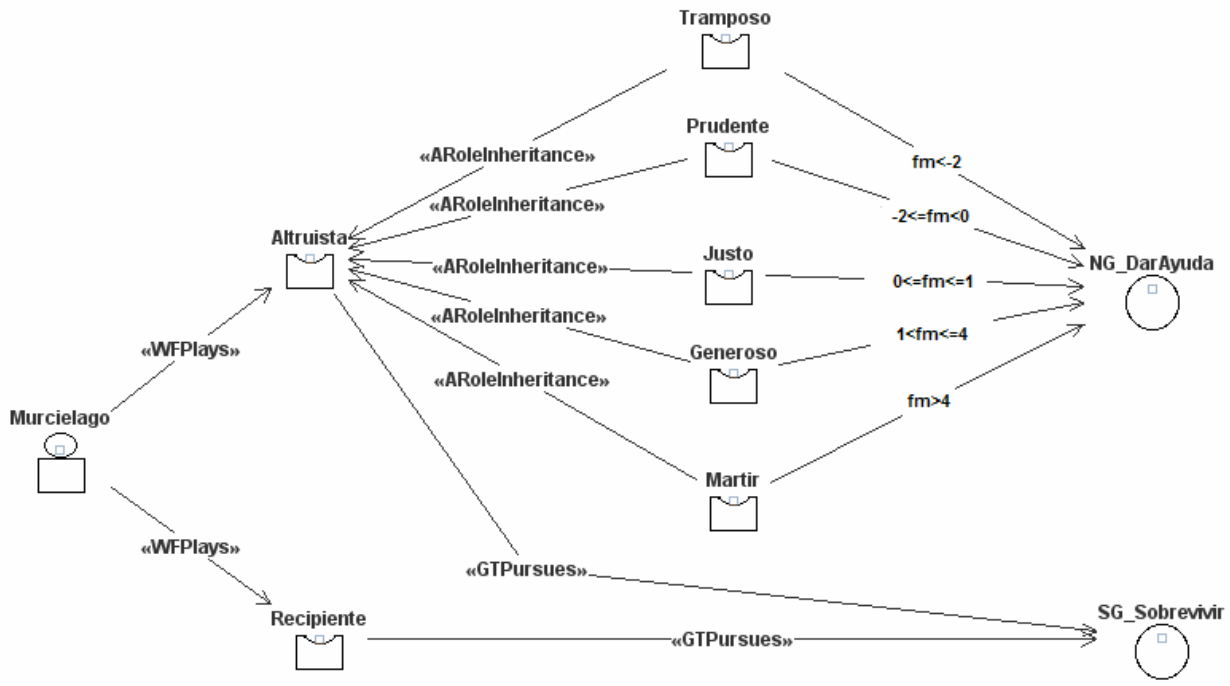

Figura 52. Perspectiva Roles y Objetivos del Modelo Altruismo con Agentes Inteligentes

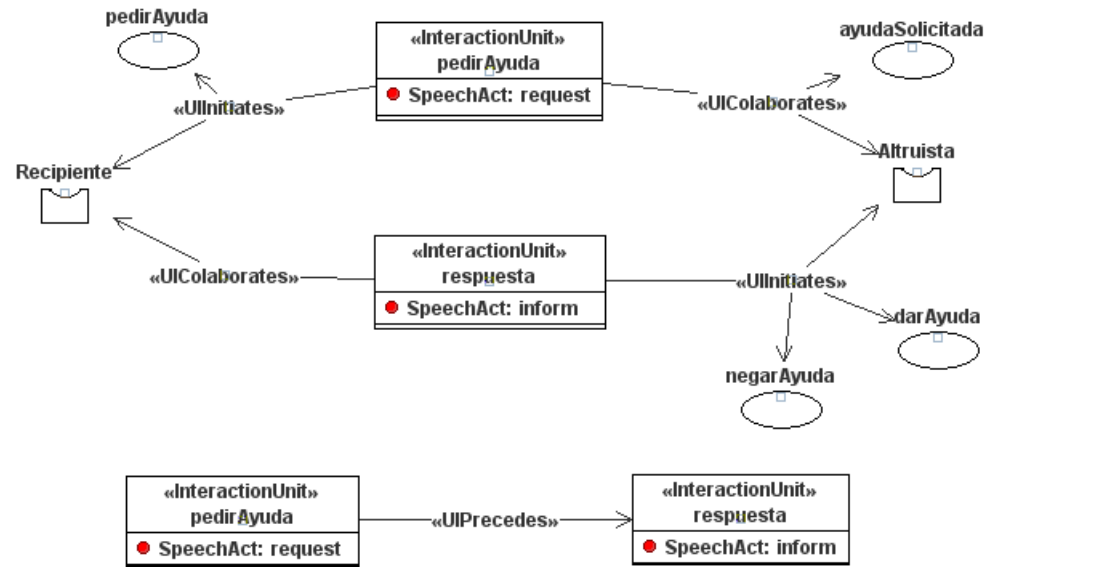

Figura 53. Perspectiva Interacción del Modelo Altruismo con Agentes Inteligentes

La perspectiva entorno se muestra en la Figura 54. El modelo de entorno determina las entidades que interaccionan con el SMA. Es una descripción de las entidades únicamente considerando los aspectos que son interesantes para la interacción de los agentes con su entorno. Para modelar

los aspectos de espacio o entornos físicos que imponen restricciones en la localización de los agentes, y para modelar el tiempo o el ciclo de actuación y percepción de los agentes se incluyen dos aplicaciones con la funcionalidad respectiva para realizar estas tareas. Las aplicaciones son accesibles a través de un API. Las aplicaciones producen eventos que pueden ser observados. Los agentes definen su percepción identificando los eventos que pueden escuchar. Los agentes actúan también sobre el entorno invocando los procedimientos o métodos ofrecidos por las aplicaciones.

En la Figura 54 se muestra cómo un agente percibe el entorno en el que se encuentran dos aplicaciones, una es la que representa la funcionalidad del espacio y la otra el tiempo. En este caso, se considera a los agentes como parte del entorno o situados en un espacio en el entorno. Esto facilita muchas tareas, pues el entorno permite a cada agente saber la lista de sus vecinos. Por ejemplo, para la interacción de los agentes por mensajes es más fácil considerar la lista de los agentes que proporciona el entorno que incluir una aplicación directorio para ubicar a los agentes. La Figura 54 muestra cómo un agente define su percepción a través de aplicaciones en el entorno. El tipo agente *Murciélago* percibe del entorno la aplicación *Espacio* y la aplicación *Tiempo*. La percepción puede concebirse como la notificación de eventos que llegan de la aplicación, p.ej. el paso de tiempo de la simulación o la etapa (día o noche) de un mismo paso de simulación. La percepción también puede concebirse como interrogación del estado de la aplicación a través de métodos, p.ej. la lista de los agentes vecinos de determinado agente.

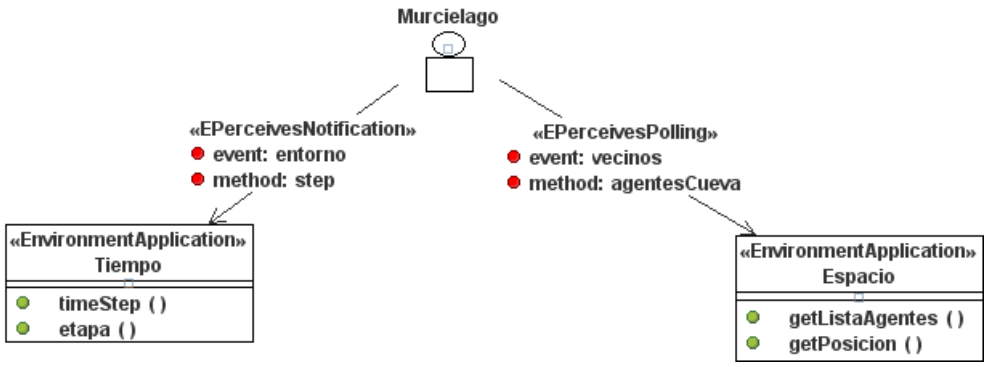

Figura 54. Perspectiva Interacción del Modelo Altruismo con Agentes Inteligentes

Finalmente, la Figura 55 muestra un diagrama de componentes que relaciona el código de las aplicaciones internas que forman parte del modelo y el código de las aplicaciones del entorno con su correspondiente implementación, el lenguaje en el que está implementado y el fichero que lo contiene. Las aplicaciones internas implementan la funcionalidad para la gestión de la *red social*, el mecanismo de *adaptación* y funcionalidad particular del caso de estudio. Las externas implementan la funcionalidad de las aplicaciones del entorno.

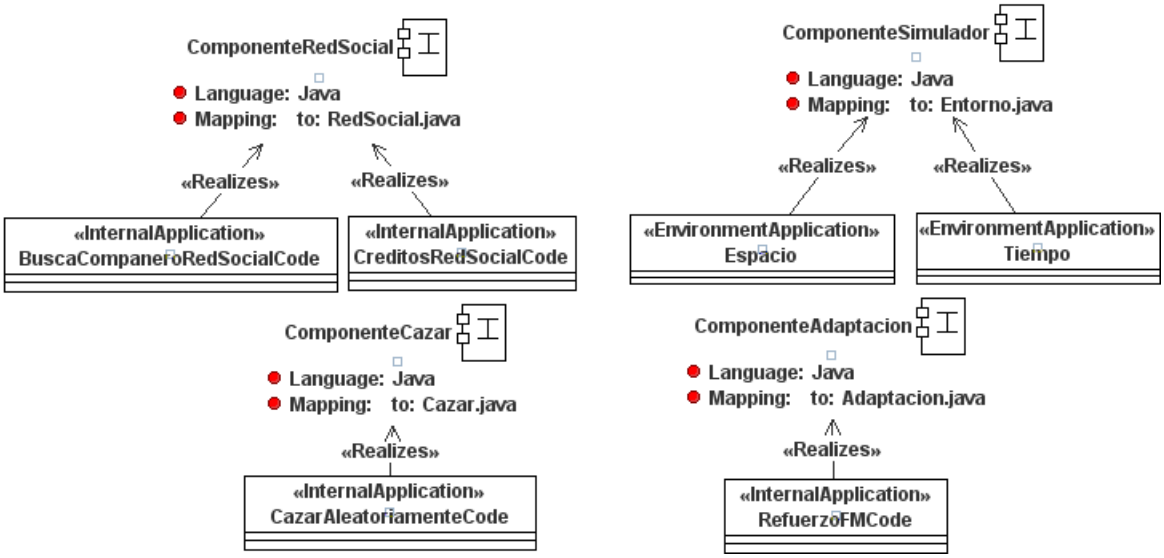

Figura 55. Diagrama de Componentes que Realizan las Respectivas Aplicaciones Internas y del Entorno del Modelo Altruismo con Agentes Inteligentes

7.3.2. Implementación

El siguiente flujo de trabajo en la fase de Ingeniería de SA es la *generación del modelo específico de la plataforma*. Éste se generó en paralelo con la especificación del modelo, se utilizaron las librerías desarrolladas para las plataformas de simulación con el modelo computacional SMA, las librerías para la generación de código y para el mecanismo de adaptación. Se realizaron varias iteraciones para refinar el modelo de ejecución. Una vez que el modelo de consideró suficientemente completo se compiló y se desplegó en las plataformas Repast y Mason. La configuración final de despliegue en cuanto al número de agentes, cuevas, etc. se realiza por facilidad a través de las interfaces que proporcionan las plataformas de simulación. La Figura 56 ilustra la interfaz de usuario que proporciona la plataforma Mason. En esta figura podemos observar que los parámetros de la simulación incluyen el número de agentes en la simulación, el número de cuevas, el número de agentes por cueva, etc.

Con el fin de poder comparar los resultados, en la versión final de la implementación se incluyó una funcionalidad que permite la reproducción de los agentes a determinada edad. La versión original del estudio no especifica cómo se lleva a cabo esta reproducción. Sin embargo, en los resultados se encuentran pruebas considerando que los agentes se reproducen y que los créditos ganados por los padres y sus características se heredan a sus crías. Así, mientras más créditos son pasados a las generaciones futuras más alta la probabilidad de supervivencia de éstas.

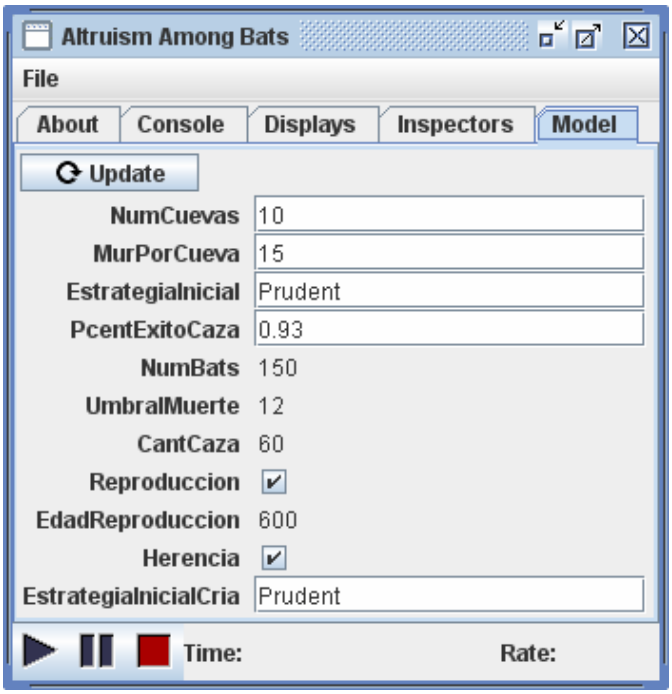

Figura 56. Interfaz de Usuario de la Plataforma Mason para Definir los Parámetros de la Simulación

7.4. Estudio de Propiedades Emergentes

Los resultados proporcionados en (Di Tosto et al. 2006) muestran que diferentes estrategias, correspondientes a diferentes patrones de relaciones entre los objetivos de los agentes, emergen en el experimento y sus diferencias se incrementan cuando las estrategias son heredadas por las crías de los agentes. En estas condiciones, la estrategia que parece ser dominante es *mártir*. Según (Di Tosto et al. 2006) estos resultados son esperados ya que esta estrategia se auto-refuerza: tan pronto como los *tramposos* empiezan a desaparecer, las donaciones incrementarán y la reciprocidad emergerá. Consecuentemente, la fuerza motivacional *fm* para perseguir el objetivo *NG_DarAyuda* aumentará en principio sin límite. En cambio, sin la herencia las estrategias no se diferenciarán y la población decrecerá gradual y lentamente. Si no se pone un límite máximo al valor *fm* creciente del objetivo *NG_DarAyuda,* la estrategia *mártir* prueba ser la más adecuada.

En la Figura 57 se muestran los resultados obtenidos al ejecutar el modelo de simulación en las plataformas Repast y Mason. La simulación se ejecutó en ambas plataformas con los mismos parámetros: 10 000 días o pasos de la simulación, una población de 150 murciélagos, 10 cuevas, cada agente desempeña la estrategia *prudente* al inicio y con el mecanismo de reproducción y herencia activado. En el lado izquierdo de la figura se encuentra la gráfica generada en la simulación con

Repast, en el lado derecho la de Mason. Basándose en los resultados presentados en el modelo original, es claro que los resultados arrojados en la simulación con Repast se alinean muy bien con los originales. En ambas simulaciones la estrategia dominante es *mártir*. La estadística arrojada por la simulación con Mason no se alinea con los resultados originales como puede observarse en la Figura 57. Nótese la diferencia de escala entre ambos ejes Y, en la gráfica de Mason, la población decrece aún cuando la reproducción está activada. Esto pudo deberse a un error en la especificación o a un error en la implementación.

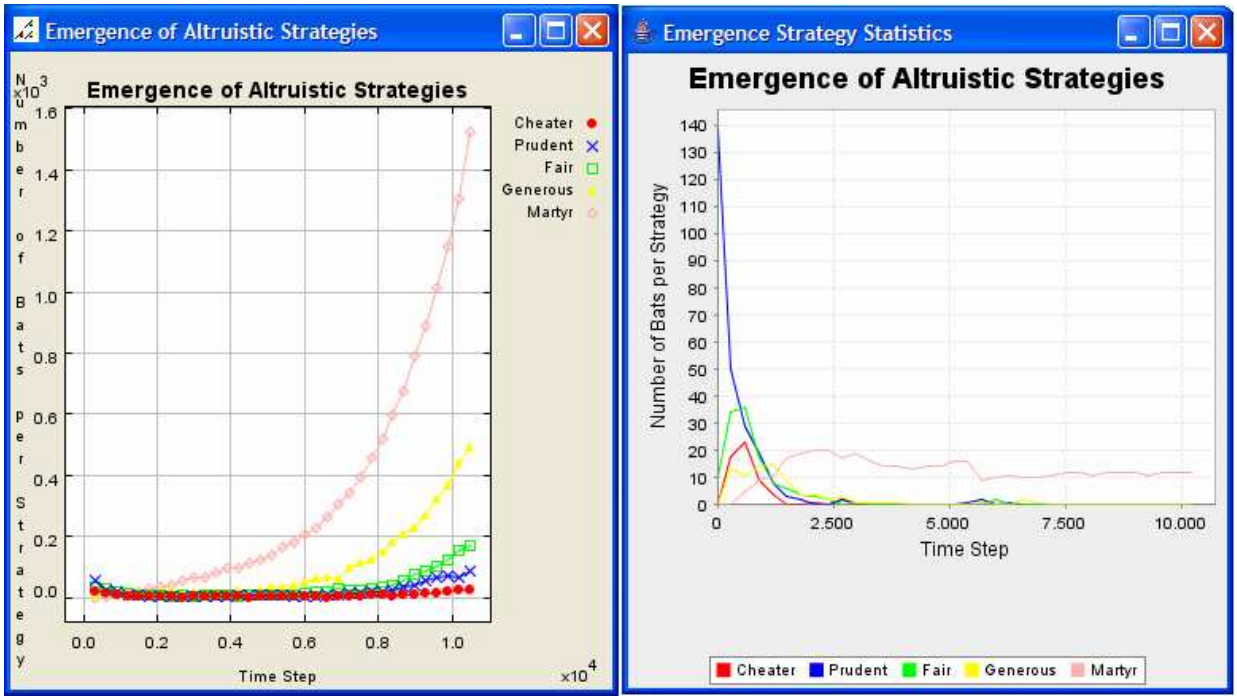

Figura 57. Emergencia de Estrategias Altruistas con Herencia. En el lado izquierdo de la figura se encuentra la gráfica estadística de Repast, en el lado derecho la de Mason. Los valores son el número de agentes por estrategia (y), y los pasos de la simulación (x). La población está compuesta de 10 cuevas de 15 agentes cada una. Cada agente es Prudente al principio

Se verificaron ambas simulaciones de forma cruzada una con la otra y aún cuando ambas implementan el mismo proceso, la implementación con Mason no produce resultados que sean suficientemente cercanos ni con el modelo original ni con los resultados de Repast. Se tiene un alto grado de confianza en la similitud de ambas implementaciones, pero una de ellas muestra diferencias significativas de los resultados originalmente publicados y la otra se alinea bastante bien al original, por lo tanto se especula que la inconsistencia podría deberse a un error introducido por el generador de números aleatorios de la simulación o por el mismo entorno de simulación. Así, se ha procedido a depurar detalladamente la implementación con Mason para encontrar posibles errores.

Esta tarea permitió descubrir algunas características de las facilidades de planificación de la plataforma Mason que introdujeron algunos sesgos en los resultados.

El mecanismo de reproducción en el modelo del altruismo que se replica en este trabajo requiere una planificación dinámica. Esto es, cuando los agentes se reproducen en el mes 20 (para modelar la fase juvenil de los agentes) sus crías son planificadas para su ejecución dinámicamente (la simulación ha iniciado y no es al principio como la de los padres). Otra característica deseable en el mecanismo de planificación para este modelo es la cuestión de la división de un mismo paso de tiempo en sub-pasos. Esto permite representar por ejemplo un día con un paso de tiempo con dos sub-pasos, uno para el día y otro sub-paso que representa la noche, y planificar agentes para su ejecución en cualquiera o en ambos sub-pasos.

Mason proporciona muchas facilidades para la planificación, incluyendo subdivisión de pasos, así como planificación dinámica. Sin embargo, durante las pruebas se encontró una cuestión que introdujo sesgo en los resultados de la simulación y que están relacionados con la planificación dinámica y la subdivisión del tiempo.

Cuando se planificaron dinámicamente las crías de los agentes durante el mes 20 para que se ejecutaran en cada paso de tiempo después de su nacimiento en los mismos sub-pasos que sus padres (de esta forma, los padres y crías juntos serían ejecutados de forma aleatoria, la intención era que fueran tratados indistintamente), el mecanismo de planificación hacía una distinción de los agentes planificados para el mismo sub-paso pero definidos en diferentes momentos de la simulación. Es decir, como los agentes padres se planificaron para su ejecución en el sub-paso *día* al inicio de la simulación y sus crías se planificaron para el mismo sub-paso pero en el paso 600 de la simulación, sucedía que los padres eran ejecutados de forma aleatoria entre ellos, y las crías también, pero nunca como una sola lista de agentes planificados para el mismo sub-paso. Parecía que cuando se planifican agentes en diferentes momentos de la simulación para un mismo sub-paso se creaba otra subdivisión.

Después de varias pruebas y de consultar con los autores de la plataforma, nos dimos cuenta que la estructura de datos *Sequence* que proporciona Mason y que se usaba para planificar a los agentes dinámicamente no es una estructura de datos dinámica, consecuentemente cuando se añadían las crías de los agentes al sub-paso, éstos eran colocados en una estructura de datos interna diferente de tal forma que en realidad se creaban varias listas de agentes según las veces que hubiera una re-planificación. Este hecho no estaba documentado en Mason y condujo a confusiones que introdujeron errores difíciles de depurar.

No obstante, contar con una segunda implementación con la cual comparar los resultados puede ayudar a encontrar estas cuestiones y descartar errores en la implementación. En la Figura 58 se puede observar la gráfica de Mason una vez que los problemas que causaban sesgo fueron resueltos y se generó de nuevo la implementación.

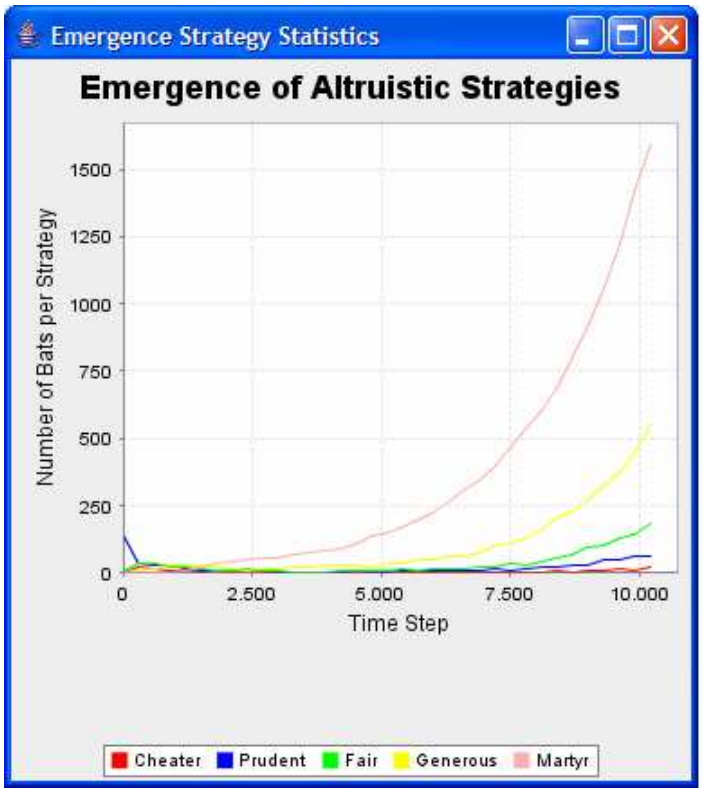

Figura 58. Emergencia de Estrategias Altruistas con Herencia. Gráfica Estadística Generada con Mason sin Sesgos de Planificación. Los valores son número de agentes por estrategia (y), y los pasos de la simulación (x). La población está compuesta de 10 cuevas de 15 agentes cada una. Cada agente es Prudente al principio

La segunda prueba que se realizó para validar la replicación fue deshabilitando el mecanismo de reproducción. En la primera prueba se implementó la reproducción con la herencia de estrategia y créditos a las crías. En esta segunda prueba, la herencia es desactivada. Los resultados se muestran en la Figura 59, sin la herencia las estrategias no se diferencian y la población decrece gradualmente. Los resultados en ambas plataformas se alinearon bien con los resultados originales.

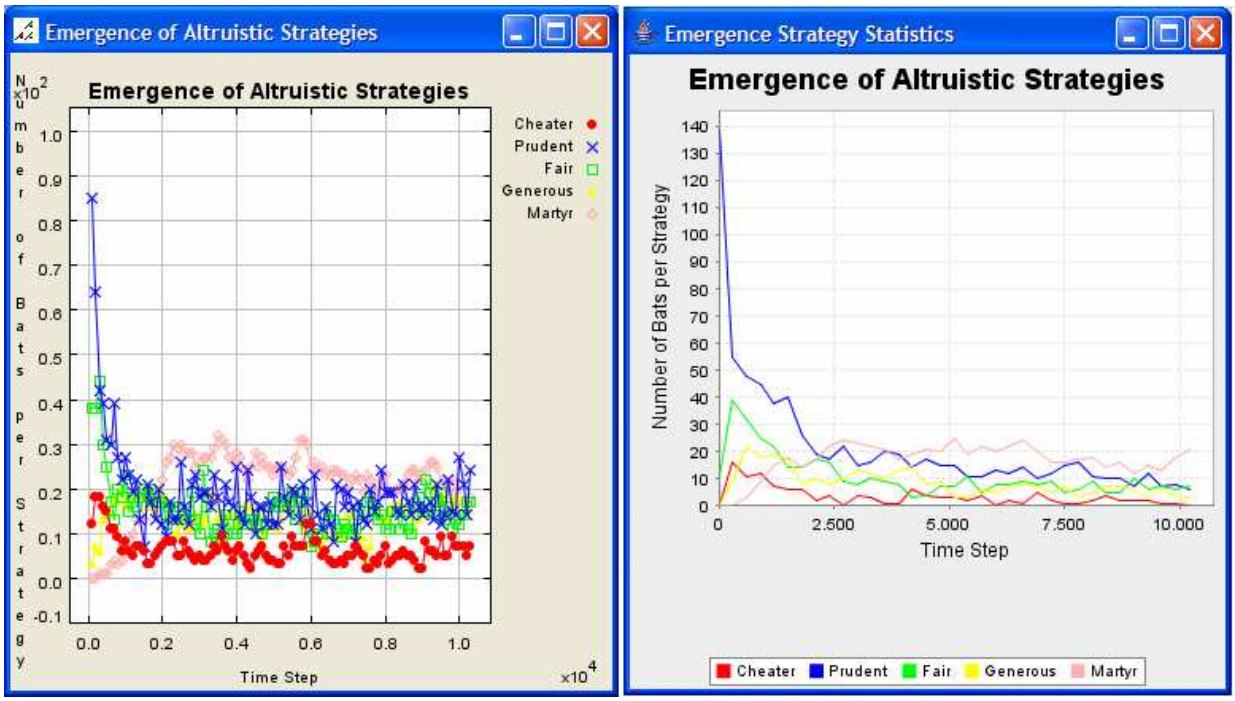

Figura 59. Emergencia de Estrategias Altruistas sin Herencia. En el lado izquierdo se muestra la gráfica estadística de Repast, en el derecho la de Mason. Los valores son el número de agentes por estrategia (y), y los pasos de la simulación (x). La población se compone de 10 cuevas de 15 agentes cada una. Cada agente es Prudente al principio

En los resultados que reportan (Di Tosto et al. 2006) se remarca el impacto de la inteligencia y de los sistemas basados en objetivos dinámicos en la propagación del altruismo. Los sistemas basados en objetivos estáticos (sin una fuerza de motivación para perseguirlos) mostraron ser peores que los sistemas simples que aplican una rutina dada. Los sistemas con objetivos dinámicos mostraron tener un efecto más estable en el altruismo, neutralizando una mayoría de agentes *trampososos*. Sin embargo, se previene que los resultados se obtuvieron bajo ciertas condiciones que pueden ayudar a comprender el altruismo. Por ejemplo, se considera que el mecanismo fundamental que está en el origen de la evolución del altruismo es la formación de grupos. Los murciélagos que viven en uni-cueva no tienen mucha oportunidad de sobrevivir compartan o no alimento. Otra cuestión a considerar, que juega un rol importante en los resultados es el número de agentes por cueva, mientras más grande sea el grupo (cueva) menos posibilidad habrá de que una red de crédito emerja. Sin la red, la reciprocidad no surgirá tampoco y los *tramposos* explotarán a los altruistas y la población desaparecerá. Un último factor a considerar son las represalias, aunque no se modelan directamente en este estudio, se considera que tienen un efecto en los resultados. Bajo todas estas condiciones, que no se consideran irrealistas, los agentes inteligentes dinámicos se desempeñan

muy bien, aún cuando los agentes *tramposos* representan la mayoría de la población. Según (Di Tosto et al. 2006) la inteligencia en lugar de prevenir puede contribuir a la propagación del altruismo, al menos cuando es flexible y puede mantener la población de *tramposos* mínima e inofensiva. Aunque también afirman los autores que con los resultados obtenidos no se puede asegurar nada con respecto a si la inteligencia les permite a los agentes seguir la misma trayectoria de aprendizaje que las especies siguieron durante su evolución, ni tampoco se puede inferir nada con respecto a las construcciones mentales subyacentes al mecanismo de compartición de alimentos de los murciélagos.

7.5. Conclusiones

En este capítulo se ha desarrollado un caso de estudio que ha permitido valorar el marco metodológico para el estudio de SA, considerando tanto los beneficios que puede aportar como las limitaciones en vista a su aplicación.

Con respecto a la especificación del modelo de simulación el lenguaje y los mecanismos de implementación cuentan con la capacidad expresiva para modelar e implementar agentes y sistemas inteligentes. Para el caso de estudio que demandaba el modelado de agentes cognitivos, dinámicos y adaptativos, el lenguaje LMSA mostró contar con las abstracciones necesarias para modelar estos aspectos. Igualmente con el modelo computacional SMA que no presentó complicación alguna para adaptarse a las necesidades del modelo de simulación.

Sin embargo, la especificación puede llegar a tener un número considerable de diagramas que pueden dificultar la gestión y la perspectiva general que se tenga del modelo. Esta dificultad puede evitarse haciendo específico el lenguaje de modelado y su correspondiente implementación. El lenguaje LMSA en su estado actual considera algunas cuestiones para la SSBA y agentes cognitivos, sin embargo para su aplicación al dominio social sería conveniente hacerlo más específico con su funcionalidad a nivel computacional correspondiente, esto ayudaría a disminuir el tamaño de las especificaciones.

Se ha visto la utilidad de la replicación en el desarrollo de modelos de SSBA pues permite aislar errores difíciles de observar, como pueden ser los introducidos por las facilidades proporcionadas por el simulador. En el caso concreto de este experimento, la replicación permitió notar parcialidades introducidas en el modelo por un error de este tipo. Con esto hemos comprobado además que no es trivial el desarrollo de programas y la utilización de librerías para la simulación. Ha habido que depurar el código de Mason con sumo cuidado para indagar la fuente de estas parcialidades.

Finalmente, un inconveniente que puede estar relacionado con la metodología, es a la hora de describir el sistema. Al principio, el usuario puede estar acostumbrado a aplicar enfoques de inge-

niería convencionales, por ejemplo el uso de un conjunto de descripciones estáticas que son responsables de hablar de la estructura del sistema, y un conjunto de descriptores de comportamiento de los elementos que componen esta estructura.

En la metodología propuesta se usan enfoques similares en el sentido de describir la estructura del sistema con componentes organizacionales. Sin embargo, para describir el comportamiento del SMA no se utilizan enfoques convencionales como los diagramas de interacción que aboca una visión del agente como una máquina de estados, lo cual se aleja del propósito y concepción inicial de los SMA (las herramientas de simulación basadas en agentes siguen este enfoque).

La metodología propuesta modela el comportamiento del sistema, en general en términos de protocolos, objetivos y tareas, es una visión más integradora de las interacciones y del modelo de intencional de los agentes. Los agentes persiguen objetivos y son éstos los que guían sus acciones. Así, en la metodología el enfoque para modelar el comportamiento del sistema se basa en el estado mental inicial de los agentes (perspectiva agente), los estados intermedios por los que atraviesa el agente (patrones de estado mental), y su evolución (perspectiva tareas, las tareas llevan al agente a evidenciar si a alcanzado un objetivo o no).

Con todo ello se espera que un usuario experto del dominio puede familiarizarse rápidamente con esta concepción de modelado si consideramos que es más cercana a la forma de concebir el comportamiento de individuos en una sociedad.

Capítulo 8.

Conclusiones Finales

En este capítulo final se recogen las principales conclusiones de la memoria: las aportaciones originales de este trabajo, las próximas líneas de investigación y el trabajo futuro.

8.1. Principales Aportaciones

La aportación principal de esta tesis ha sido una metodología para el estudio de SA. La metodología plantea una guía para el desarrollo y análisis de SA como sociedades de agentes inteligentes. La metodología se cimienta en la ingeniería de desarrollo basado en modelos que permite trabajar en un nivel de abstracción elevado independiente de las tecnologías de implementación subyacentes. La infraestructura necesaria para la aplicación de la metodología se propone como un marco. Éste incluye un lenguaje de modelado de sociedades artificiales (LMSA), que considera aspectos de intencionalidad y racionalidad de los agentes y su modelo computacional correspondiente, un mecanismo para aspectos de adaptabilidad de los agentes y un módulo para la transformación automática de los modelos a código de simulación.

La hipótesis con la que iniciamos este trabajo fue que las prácticas actuales para el estudio de SA podían mejorarse en tres aspectos: la metodología de desarrollo de modelos basados en agentes, el nivel de desarrollo del concepto de agente con el que se pueden modelar e implementar sistemas sociales y el alto grado de conocimiento de programación que requiere el experto del dominio social para el desarrollo de estas sociedades.

Algunas metodologías existentes se concentran en las etapas de experimentación y eliden el desarrollo de SMA con los que se modelan las SA. Otras que si consideran esta actividad se basan en modelos de agentes sencillos, con lo cual, el proceso de elaboración que proponen es incompleto cuando se trata de modelar sociedades de agentes inteligentes con patrones mentales que afectan o restringen el desarrollo de la sociedad. Unas más avanzadas consideran la necesidad de un modelo de SMA más desarrollado, sin embargo, no definen como elaborarlo.

En cuanto a las herramientas de implementación, en su mayoría están orientadas a lenguajes de programación que proporcionan la libertad de definir el modelo de agente adecuado a cada problema. Otras contemplan un nivel más elevado de especificación pero con modelos de agentes extremadamente sencillos.

La propuesta de este trabajo trata de facilitar el modelado de SA incrementando el nivel de abstracción utilizado, para permitir el análisis de patrones sociales emergentes a nivel del modelo.

Para ello se aborda la necesidad de extender los conceptos de agentes puramente reactivos sobre los que se sustentan las metodologías y herramientas actuales a conceptos de agentes inteligentes que facilitaran la especificación de modelos de sistemas sociales. Como las sociedades pueden estar formadas por individuos que toman decisiones y que actúan, hacía falta utilizar un concepto de agente más desarrollado, con características sociales e intencionales, en línea con las propuestas de la ingeniería del software orientada a agentes. Además, siguiendo con este mismo razonamiento, fue necesario considerar un lenguaje para el modelado abstracto de estos agentes inteligentes y un mecanismo que facilitara su representación computacional y su posterior simulación.

Así, para la realización de las técnicas que abordarían los aspectos anteriores se planteó el uso de una metodología de las existentes en la ingeniería del software orientada a agentes, concretamente INGENIAS. La justificación de su adecuación para realizar estas técnicas se basa en las características del modelo de SMA que adopta y su filosofía de desarrollo basada en modelos. Con ello, fue posible considerar aspectos micro de un sistema social como el comportamiento intencional de los agentes y aspectos macro, como la estructura organizativa y su dinámica.

El trabajo de investigación desarrollado para integrar esta metodología al estudio de SA, cubriendo las necesidades planteadas anteriormente, condujo a las siguientes aportaciones:

1. El lenguaje **LMSA** para el modelado visual de SA. Éste se definió con un conjunto de abstracciones del dominio de la simulación social que se incorporaron al lenguaje de modelado de SMA INGENIAS. El lenguaje LMSA permite la especificación de modelos en una forma menos ambigua que el lenguaje natural usado en las disciplinas sociales. Esta especificación se integra naturalmente al proceso de desarrollo propuesto. El lenguaje puede extenderse para incluir nuevos conceptos del dominio social. Finalmente, el lenguaje facilita la identificación y análisis de procesos y patrones sociales emergentes en términos de los elementos de la especificación.

2. Un **Modelo Computacional de SMA para Simulación Social**. Los elementos de este modelo corresponden e implementan los elementos del modelo conceptual de SMA del lenguaje LMSA, con el fin de preservar la semántica y que la aplicación resultante sea funcionalmente equivalente a dicho modelo. El modelo computacional considera por lo tanto un modelo de SMA de agentes inteligentes. Éste se coloca sobre las plataformas de simulación Repast y Mason para dotarlas de un modelo de agente más desarrollado que los que éstas adoptan. La correspondencia entre los elementos de ambos niveles de abstracción permite la interpretación de resultados en términos de los elementos del modelo conceptual.
3. Una **Arquitectura de Adaptabilidad por Refuerzo**. Se diseñó para proporcionar más realismo al modelado de individuos en un sistema social complejo. La arquitectura dota a los agentes de aptitudes para modificar dinámicamente su comportamiento de acuerdo a algún refuerzo. Este comportamiento se logra mediante la motivación de los agentes para perseguir determinados objetivos. La consecuencia de este refuerzo sobre la motivación es que un agente individual puede adaptar sus aptitudes y puede exhibir una especialización de roles.
4. Un **Módulo para la Generación Automática de Código y el Análisis de SA**. Este módulo se ocupa de transformar el modelo en código fuente automáticamente. Se construye con la interfaz que proporciona INGENIAS para recorrer las especificaciones, seleccionar la información del modelo que se quiere implementar y colocarla en los prototipos diseñados para las plataformas de simulación Repast y Mason en las que será posteriormente ejecutado. Esta estrategia de desarrollo basado en modelos permite la replicación para comparar resultados y para estudiar los efectos que pueden tener las facilidades de distintos entornos de simulación sobre los resultados de la simulación. Para el análisis, los resultados se presentan directamente en los entornos de simulación, de esta manera los diagramas de resultados que proporciona el simulador son fácilmente interpretables para identificar patrones sociales por el usuario en términos de los elementos del modelo. Para ello se consideran esenciales las dos primeras aportaciones, es decir que se cuente con una especificación abstracta y que sus elementos se hayan correspondido adecuadamente con elementos del código del modelo en el simulador. Además, estas tres aportaciones en conjunto permiten aislar al experto del dominio de aspectos de implementación y fomentan su concentración en el análisis de resultados.
5. La M**etodología para el Estudio de SA**. Sirve de guía para el desarrollo de SA como SMA y su integración a un proceso más amplio de experimentación. Su filosofía de desarrollo basado en modelos es su propuesta para facilitar esta tarea. Define con SPEM un proceso para la elaboración de SMA de agentes intencionales y racionales, incluyendo su diseño y la transformación de su especificación a un modelo de simulación ejecutable. La metodolo-

gía define un proceso iterativo e incremental para el estudio de SA con tres fases. La *fase de identificación del objeto de estudio*, la *fase de ingeniería de SA*, y la *fase de experimentación*. Para cada fase define flujos de trabajo. Así se identifican respectivamente los siguientes: *definición del sistema*, *generación del modelo de diseño* y *generación del modelo específico de la plataforma*, y *estudio de propiedades emergentes*. Para cada flujo de trabajo, la metodología define las actividades y los roles que participan en cada actividad, los artefactos que produce, y las herramientas utilizadas. La definición de roles en este ciclo permite que cada uno se centre en las actividades que mejor realiza. Además, delimita las responsabilidades y mejora la comunicación entre los roles. Con esto, los expertos del dominio pueden enfocar su esfuerzo en el modelado y evaluación de resultados.

Con estas aportaciones se mejora el estudio de SA en dos sentidos, uno en el modelado y desarrollo de agentes que exhiben características racionales, intencionales y de adaptabilidad, y otro, en el análisis de la emergencia de los efectos imprevisibles y agregados de estas sociedades.

8.2. Líneas de Investigación Abiertas

La dificultad más destacable de esta propuesta está relacionada con la especificación y la generación automática de código. El modelado de un SMA contempla muchos aspectos que deben considerarse y su especificación puede llegar a ser bastante compleja. Con un lenguaje visual y una guía para su desarrollo la dificultad logra aminorarse, pero aún prevalece el nivel de detalle excesivo que se requiere para la generación automática de código y ciertos aspectos qué la guía no logra cubrir debido a su carácter general. Para ello, se plantea definir lenguajes de modelado para dominios de aplicación específicos, por ejemplo, para una teoría social específica o un dominio de estudio concreto, que considere aspectos concretos de SMA relevantes para su aplicación. Esto reduce no solamente la complejidad de la especificación del modelo, sino también el nivel de detalle que debe especificarse para su implementación.

Otro aspecto que puede mejorarse es ampliar la librería de mecanismos de auto-organización. Existen diversos mecanismos que son más adecuados para situaciones diversas. El mecanismo de adaptabilidad por refuerzo fue más adecuado para modelar la auto-organización que presentan los individuos cuando aprenden de experiencias pasadas. Sin embargo, sería interesante incluir otros mecanismos para modelar la auto-organización, por ejemplo, por medio del comportamiento cooperativo de los individuos cuando interaccionan localmente con otros. Un método que se ha considerado incluir es el basado en la teoría AMAS (Capera et al. 2003; Georgé et al. 2003). Este método permitiría modelar individuos con la capacidad de auto-organizarse, reorganizando sus interacciones locales con otros agentes y el entorno, cuando reconocen una situación no-

cooperativa. Para ello, se basan en su conocimiento, en las representaciones que tienen de otros agentes y en la tarea individual que tienen que realizar.

A propósito del módulo para el análisis de SA, actualmente no se proporciona otra forma para la interpretación de resultados que las mismas que proporcionan las plataformas de simulación. Hasta ahora, bajo el marco de experimentación que se ha realizado durante este estudio ha sido suficiente. Sin embargo, en un marco más amplio pueden requerirse otras herramientas de ayuda al análisis, por ejemplo, para el estudio de sensibilidad del modelo con respecto a los parámetros iniciales.

Con respecto a la generación automática de código, se ha experimentado con dos plataformas de simulación ampliamente utilizadas en el dominio de la simulación social. La propuesta de ayuda al desarrollo que hace el marco metodológico no está limitada a estos dos entornos, por lo que es factible y deseable incluir nuevos entornos para ampliar la experimentación.

Finalmente, esta propuesta ha sido validada con experimentos en el seno de nuestro grupo de investigación. Queda pendiente una experimentación más amplia con otros proyectos, y a más largo plazo la posible exportación de las técnicas propuestas a otros dominios de aplicación.

8.3. Publicaciones Relacionadas con la Tesis

A lo largo del trabajo de investigación que ha implicado el desarrollo de esta tesis se han producido una serie de publicaciones que se detallan a continuación en orden cronológico:

ARTÍCULOS EN CONGRESOS

1. (Sansores et al. 2004). Sansores, C., J. Pavón, A. López-Paredes (2004). *Towards a Framework for ABSS on the Grid*. Proceedings of the Second Conference of the European Social Simulation Association (ESSA 2004), ISBN 84-688-7964-9, Valladolid, Spain, September 16-19, 2004.
2. (Sansores y Pavón 2004a). Sansores, C., J. Pavón (2004a). *Agents on Social Sciences: ABSS Case Study*. Proceedings of the Fifth Iberoamerican Workshop on Multi-agent Systems (IBERAGENTS 2004), Puebla, México, November 23, 2004a.
3. (Sansores y Pavón 2004b). Sansores, C., J. Pavón (2004b). *A Framework for Agent Based Social Simulation*. Proceedings of the Second European Workshop on Multi-Agent Systems (EUMAS 2004), Barcelona, Spain, December 16-17, 2004b.
4. (Sansores y Pavón 2004c). Sansores, C., J. Pavón (2004c). *MAS Scalability: ABSS on the Grid Case Study*. Proceedings of the Third International Workshop on Practical Applica-

tions of Agents and Multi-agent Systems (IWPAAMS 2004), Burgos, Spain, October 13-15, 2004c.

5. (Sansores et al. 2005). Sansores, C., J. Pavón, J. Gómez-Sanz (2005). *Visual Modeling for Complex Agent-Based Simulation Systems*. In: J. Sichman and L. Antunes (Eds.): Multi-Agent-Based Simulation, Sixth International Workshop on Multi-Agent-Based Simulation (MABS 2005), Revised and Invited Papers (2006), Lecture Notes in Computer Science, Vol. 3891. Springer-Verlag, Utrech, Netherlands, July 25, 2005, pp. 174-189.
6. (Sansores y Pavón 2005a). Sansores, C., J. Pavón (2005a). *Agent-Based Simulation Replication: a Model Driven Architecture Approach*. In: A. Gelbukh et al (Eds.): Fourth Mexican International Conference on Artificial Intelligence (MICAI-2005), Lecture Notes in Artificial Intelligence, Vol. 3789. Springer-Verlag, Monterrey, México, November 14-18, 2005a, pp. 244-256.
7. (Sansores y Pavón 2005b). Sansores, C., J. Pavón (2005b). *Agent Based Modeling of Social Complex System (Thesis Proposal)*. In: R. Marín et al (Eds.): 11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2005), Revised and Invited Papers (2006), Current Topics in Artificial Intelligence, Lecture Notes in Artificial Intelligence, Vol. 4177. Springer-Verlag, Santiago de Compostela, Spain, November 14-15, 2005b, pp. 99-102.
8. (Sansores y Pavón 2005c). Sansores, C., J. Pavón (2005c). *Agent Based Simulation for Social Systems: from Modeling to Implementation*. In: R. Marín et al (Eds.): 11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2005), Revised and Invited Papers (2006), Current Topics in Artificial Intelligence, Lecture Notes in Artificial Intelligence, Vol. 4177. Springer-Verlag, Santiago de Compostela, Spain, November 14-15, 2005c, pp. 79-88.
9. (Pavón et al. 2006). Pavón, J., M. Arroyo, S. Hassan, C. Sansores (2006). *Simulación de Sistemas Sociales con Agentes Software*. Actas del Campus Multidisciplinar en Percepción e Inteligencia, CMPI 2006, Vol. I. Albacete, Spain, July 10-14, 2006, pp. 389-400.

ARTÍCULOS EN REVISTAS

1. (Sansores y Pavón 2005d). Sansores, C., J. Pavón (2005d). *Simulación Social Basada en Agentes*. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial "Aplicaciones de la Tecnología de Agentes", ISSN 1137-3601, Vol. 9(25). pp. 71-78.
2. (Pavón et al. 2007a). Pavón, J., M. Arroyo, S. Hassan, C. Sansores (2007a). *Agent Based Modelling and Simulation for the Analysis of Social Patterns*. In: Pattern Recognition Let-

ters, Special Issue on Pattern Recognition in Interdisciplinary Perception and Intelligence, Elsevier Science, 2007a. (Aceptado para Publicación).

3. (Pavón et al. 2007b). Pavón, J., C. Sansores, J. Gómez-Sanz (eds.) (2007b). Modelling and Simulation of Social Systems with INGENIAS. International Journal of Agent Oriented Software Engineering, 2007b, Inderscience Publishers. (En Revisión).

CAPÍTULOS DE LIBRO

1. (Pavón y Sansores 2006). Pavón, J., C. Sansores (2006). *Using INGENIAS for Visual Modeling of Complex Agent Based Simulation Systems*. In: Agent-Based Modelling in Natural Resource Management, edited by A. López-Paredes and C. Hernández, Pearson Education, Madrid, 2006, pp. 173-207.

Bibliografía

AgentSheets (2004). *AgentSheets: Agent-based end-user development*. 2004. Available from: http://www.agentsheets.com

Alexander, J. C., B. Giesen (1987). *From reduction to linkage: The long view of the micro-macro link*. In: The micro-macro link, edited by J. C. Alexander et al, University of California Press, Berkeley, 1987, pp. 1-42.

Archer, M. S. (1995). Realist social theory: the morphogenetic approach. 1995, Cambridge University Press, Cambridge; New York.

Ascape (2000). *Ascape: Agent based research*. 2000. Available from: http://www.brook.edu/es/dynamics/models/ascape

Axelrod, R. (1997a). *Advancing the Art of Simulation in the Social Sciences*. In: R. Conte et al (Eds.): Simulating Social Phenomena, Lecture Notes in Economics and Mathematical Systems, Vol. 456. Berlin Springer, 1997a, pp. 21-40.

Axelrod, R. (1997b). *The Dissemination of Culture: A Model with Local Convergence and Global Polarization*. Journal of Conflict Resolution, Vol. 41(2). pp. 203-226.

Axelrod, R. M. (1997c). The complexity of cooperation: agent-based models of competition and collaboration. In Princeton studies in complexity. 1997c, Princeton University Press, Princeton, N.J.

Axtell, R. (2000). *Why Agents? On the Varied Motivations for Agent Computing in the Social Sciences*. Working Paper No. 17, November 2000, 2000.

Axtell, R., R. Axelrod, J. M. Epstein, M. D. Cohen (1996). *Aligning Simulation Models: A Case Study and Results*. Computational and Mathematical Organization Theory, Vol. 1(2). pp. 123-141.

Axtell, R., R. M. Axelrod, J. M. Epstein, M. D. Cohen (1997). *Replication of Agent-Based Models, Aligning Simulation Models: A Case Study and Results*. In: The Complexity of Cooperation, Princeton University Press, Princeton, N.J., 1997, pp. 183-205.

Bertalanffy, L. v. (1968). General system theory; foundations, development, applications. 1968, G. Braziller, New York.

Beydeda, S. (2005). Model-Driven Software Development, edited by S. Beydeda et al. 2005, Springer, Berlin.

Bonabeau, E., M. Dorigo, G. Theraulaz (1999). Swarm intelligence: from natural to artificial systems. 1999, Oxford University Press, New York.

Bratley, P., B. L. Fox, L. E. Schrage (1987). A Guide to Simulation. 2nd ed. 1987, Springer-Verlag Inc., New York, USA.

Burkhart, R., M. Askenazi, N. Minar (2000). *Swarm Release Documentation*. 2000. Available from: http://www.santafe.edu/projects/swarm/swarmdocs/set/set.html

Capera, D., J.-P. George, M.-P. Gleizes, P. Glize (2003). *The AMAS theory for complex problem solving based on self-organizing cooperative agents*. In: I. C. Society (Eds.): 1st International workshop on Theory and Practice of Open Computational Systems (TAPOCS) at IEEE 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2003)IEEE Computer Society, 9-11 June 2003, 2003, pp. 383-388.

Carrington, P. J., J. Scott, S. Wasserman (2005). Models and methods in social network analysis. In Structural analysis in the social sciences; 27. 2005, Cambridge University Press, Cambridge; New York.

Castelfranchi, C. (1997). *Principles of Individual Social Action*. In: Contemporary Action Theory, edited by R. Tuomela and G. Hinitikka, Kluwer, Dordrecht, 1997, pp. 24-29.

Castelfranchi, C. (1998a). *Modelling social action for AI agents*. Artificial Intelligence, Vol. 103(1-2). pp. 157.

Castelfranchi, C. (1998b). *Simulating with Cognitive Agents: The Importance of Cognitive Emergence*. In: Proceedings of the First International Workshop on Multi-Agent Systems and Agent-Based Simulation. 1998b, Springer-Verlag.

Castelfranchi, C., F. Paglieri (2007). *The Role of Beliefs in Goal Dynamics: Prolegomena to a Constructive Theory of Intentions*. Synthese, Vol. 155(2). pp. 237-263.

Casti, J. L. (1994). Complexification: Explaining a Paradoxical World through the Science of Surprise. 1994, HarperCollins, New York.

Casti, J. L., A. Karlqvist (1986). Complexity, language, and life: mathematical approaches. In Biomathematics; v. 16. 1986, Springer-Verlag, Berlin; New York.

Caws, P. (2000). Structuralism: A Philosophy for the Human Sciences. 2000, Humanity Books, New York.

Cilliers, P. (1998). Complexity and postmodernism: understanding complex systems. 1998, Routledge, London; New York.

Coleman, J. S. (1987). *Microfoundations and Macrosocial Behavior*. In: The Micro-Macro Link, edited by B. G. Jeffrey C. Alexander, Richard Münch, and Neil J. Smelser, University of California Press, Berkeley, 1987, pp. 153-173.

Coleman, J. S. (1990). Foundations of Social Theory. 1990, Harvard University Press, Cambridge, MA.

Collier, N., T. Howe, M. North (2003). *Onward and Upward: The Transition to Repast 2.0.* First Annual North American Association for Computational Social and Organizational Science ConferenceElectronic Proceedings, Pittsburgh, PA USA, June 2003, 2003.

Conte, R. (1999). *Social Intelligence Among Autonomous Agents*. Computational and Mathematical Organization Theory, Vol. 5(3). pp. 203.

Conte, R. (2000). *The Necessity of Intelligent Agents in Social Simulation*. Advances in Complex Systems, ISSN 0219-5259, Vol. 3(1). pp. 19-39.

Conte, R., C. Castelfranchi (1995). Cognitive and Social Action. 1995, UCL Press, London.

Conte, R., B. Edmonds, S. Moss, R. K. Sawyer (2001). *Sociology and Social Theory in Agent Based Social Simulation: A Symposium*. Computational & Mathematical Organization Theory, Vol. 7(3). pp. 183-205.

Conte, R., N. Gilbert, J. Sichman (1998). *MAS and Social Simulation: A Suitable Commitment*. Multi-Agent-Based Simulation, First International Workshop, MABS 1998, Paris, France, 1998, pp. 97-107.

Cormas (2004). *Cormas: Natural resources and agent-based simulations*. 2004. Available from: http://cormas.cirad.fr/indexeng.htm

CSCS (2005). *Center for the Study of Complex Systems*. 2005. Available from: http://www.cscs.umich.edu/

CWM (2001). *Common Warehouse Metamodel Specification*. OMG documents: ad/01-02-(01,02,03), 2001.

Darwin, C. (1859). On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life. 1859, John Murray: Printed by W. Clowes and Sons, London.

Di M. Serugendo, G., M.-P. Gleizes, A. Karageorgos (2006). *Self-Organization and Emergence in MAS: An Overview*. Informatica, Vol. 30(1). pp. 45-54.

Di Tosto, G., M. Paolucci, R. Conte (2006). *Altruism Among Simple and Smart Vampires*. International Journal of Cooperative Information Systems(Special Issue).

Drogoul, A., D. Vanbergue, T. Meurisse (2002). *Multi-agent Based Simulation: Where Are the Agents?* In: J. S. Sichman et al (Eds.): Multi-Agent-Based Simulation II: Third International Workshop, MABS 2002, Lecture Notes in Computer Science, Vol. 2581. Springer, Bologna, Italy, July 15-16, 2002, pp. 1-15.

Dumouchel, P., J. P. Dupuy (1983). L'Autorganization. De la Physique au Politique. 1983, Editions du Seuil, Paris.

Dupuy, J. P. (1992). Introduction aux Sciences Sociales. Logique des Pheénomès Collectifs. 1992, Editions Ellipses, Paris.

Durkheim, E. (1964). The rules of sociological method. 8th ed. 1964, Free Press, New York.

Elzas, M. S., B. P. Zeigler, T. I. Oren (1989). Modelling and Simulation Methodology: Knowledge Systems Paradigms. 1989, Elsevier Science, New York, USA.

Epstein, J. M., R. Axtell (1996). Growing artificial societies: social science from the bottom up. 1996, Brookings Institution Press MIT, Cambridge, MA.

Ferber, J. (1999). Multi-Agent Systems: an Introduction to Distributed Artificial Intelligence. 1999, Addison-Wesley Longman, Harlow.

Fisher, R. A. (1958). The genetical theory of natural selection. [2d rev. ed. In Dover books on science; S466. 1958, Dover Publications, New York.

Fishwick, P. A. (1995a). *Computer Simulation: The Art and Science of Digital World Construction*. IEEE Potentials. pp. 24-27.

Fishwick, P. A. (1995b). Simulation Model Design and Execution. 1995b, Prentice Hall.

Forrester, J. W. (1961). Industrial dynamics. 1961, MIT Press, Cambridge, MA.

Forrester, J. W. (1971). World dynamics. 1971, Wright-Allen Press, Cambridge, Mass.

Foxwell, H. (1999). *Java 2 Software Development Kit*. Linux Journal.

Galán, J. M. (2007). *Evaluación Integradora de Politicas de Agua: Modelado y Simulación con Sociedades Artificiales de Agentes*. In: R. del Olmo and A. López-Paredes (Advisors): Departamento de Ingeniería Civil. 2007, Universidad de Burgos.

Galán, J. M., T. E. Downing, A. López-Paredes, C. Warwick (2003). *Rigour and Reliability in Agent-Based Social Simulation Through Replication*. In: Online Proceedings of the First Conference of the European Social Simulation Association. 2003: Groningen, The Netherlands.

Galán, J. M., L. R. Izquierdo (2005). *Appearances Can Be Deceiving: Lessons Learned Re-Implementing Axelrod's Evolutionary Approach to Norms*. Journal of Artificial Societies and Social Simulation, Vol. 8(3).

Gamma, E., R. Helm, R. Johnson, J. Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. In Addison-Wesley Professional Computing Series. 1995, Addison-Wesley, Boston, Mass.

Gaylord, R., L. J. D'Andria (1998). Simulating Society. A *Mathematica* Toolkit for Modeling Socioeconomic Behavior. 1998, Springer-Verlag, New York.

Gell-Mann, M. (1994). The quark and the jaguar: adventures in the simple and the complex. 1994, W.H. Freeman and Co., New York.

Genesereth, M. R., S. P. Ketchpel (1994). *Software Agents*. Communications of the Association for Computing Machinery. pp. 48-53.

Georgé, J.-P., M.-P. Gleizes, P. Glize (2003). *Conception de systèmes adaptatifs à fonctionnalité émergente: la théorie Amas*. Revue IA, Vol. 17(4). pp. 591-626.

Gilbert, N. (1996). *Environments and Languages to Support Social Simulation*. In: K. G. Troitzsch et al (Eds.): Social Science MicrosimulationBerlin Springer, Schloss Dagstuhl, 1996, pp. 457-458.

Gilbert, N. (1999). *Simulation: A New Way of Doing Social Science*. American Behavioral Scientist, ISSN 0002-7642, Vol. 00042(00010). pp. 1485-1488.

Gilbert, N. (2002). *Varieties of emergence in social simulation*. In: Social agents: Ecology, exchange, and evolution, edited by C. M. Macal and D. L. Sallach, Argone National Laboratory, Chicago, IL, 2002, pp. 41-50.

Gilbert, N., R. Conte (1995). Artificial societies: the computer simulation of social life. 1995, UCL Press, London.

Gilbert, N., J. Doran (1994). Simulating societies: the computer simulation of social phenomena. 1994, UCL Press, London.

Gilbert, N., U. Mueller, R. Suleiman, K. G. Troitzsch (1997). *Dagstuhl Seminar on Social Science Microsimulation: Tools for Modeling, Parameter Optimization, and Sensitivity Analysis*. Dagstuhl-Seminar-Report 177, 1997.

Gilbert, N., P. Terna (2000). *How to build and use agent-based models in social science*. Mind & Society, Vol. 1. pp. 55-72.

Gilbert, N., K. G. Troitzsch (1996). Simulation for the Social Scientist. 1996, Open University Press, Buckingham, U.K.

Gilbert, N., K. G. Troitzsch (1999). Simulation for the Social Scientist. 1999, Open University Press, Buckingham, U.K.

Goldspink, C. (2000). *Modelling Social Systems as Complex: Towards a Social Simulation Meta-model*. Journal of Artificial Societies and Social Simulation, Vol. 3(2).

Goldspink, C. (2002). *Methodological Implications of Complex Systems Approaches to Sociality: Simulation as a foundation for knowledge*. Journal of Artificial Societies and Social Simulation, Vol. 5(1).

Goldstein, J. (1999). *Emergence as a Construct: History and Issues*. Emergence, Vol. 1(1). pp. 49.

Harvey, B. (1997). *Computer Science Logo Style*. 1997, MIT Press: Boston, MA.

Homans, G. C. (1964). *Commentary*. Sociological Inquiry, Vol. 34. pp. 221-231.

Huber, J. (1991). Macro-micro linkages in sociology. In American Sociological Association presidential series. 1991, Sage Publications, Newbury Park, Calif.

IDK (2004). *INGENIAS Development Kit*. 2004. Available from: http://grasia.fdi.ucm.es/ingenias

Jennings, N. R. (2000). *On agent-based software engineering*. AI Journal, Vol. 117. pp. 277-296.

Jennings, N. R., M. J. Wooldridge (2000). Agent-Oriented Software Engineering. In Handbook of Agent Technology, edited by J. Bradshaw. 2000, AAAI/MIT Press.

Kluegl, F. (2001). Multi-Agent Simulation - Concepts, Tools, Application. 2001, Addison Wesley.

Klügl, F. (2001). Multi-Agent Simulation - Concepts, Tools, Application. 2001, Addison Wesley.

Klügl, F., R. Herrler, C. Oechslein (2004). *From Simulated to Real Environments: How to Use SeSAm for Software Development*. Lecture Notes in Computer Science, Vol. 2831. 2004, pp. 13-24.

Knorr-Cetina, K., A. V. Cicourel (1981). Advances in social theory and methodology: toward an integration of micro- and macro-sociologies. 1981, Routledge & Kegan Paul, Boston.

Konolige, K. (ed.) (1992). Autoepistemic Logic. HandBook of Logic in Artificial Intelligence and Logic Programming, Vol. III, 1992, Oxford, Clarendon Press.

Kontopoulos, K. M. (1993). The logics of social structure. In Structural analysis in the social sciences; 6. 1993, Cambridge University Press, Cambridge; New York.

Langton, C. (1989). Artificial Life. In Santa Fe Institute Studies in the Sciences of Complexity, Proceedings. Vol. 6. 1989, Addison-Wesley, Redwood City, CA.

Langton, C. G. (1990). *Computation at the Edge of Chaos: Phase-Transitions and Emergent Computation*. Physica D: Nonlinear Phenomena, Vol. 42(1-3). pp. 12-37.

Langton, C. G., C. Taylor, J. D. Farmer, S. Rasmussen (eds.) (1992). Artificial life II: Proceedings of the Workshop on Artificial Life held February, 1990 in Santa Fe, New Mexico. SFI Studies in the Sciences of Complexity, 1992, Redwood City, CA, Addison-Wesley.

López Paredes, A., C. Hernández, J. Pajares Gutiérrez (2002). *Towards a New Experimental Socio-economics Complex Behaviour in Bargaining*. Journal of Socioeconomics, Vol. 31. pp. 423-429.

Ludewig, J. (2003). *Models in Software Engineering - An Introduction*. Software and System Modeling, Vol. 2(1). pp. 5-14.

Luke, S., G. C. Balan, L. Panait, C. Cioffi-Revilla, S. Paus (2003). *MASON: A Java Multi-Agent Simulation Library*. Proceedings of the Agent 2003 Conference, Chicago, IL, USA., 2003.

Lutz, M., D. Ascher (1999). Learning Python. 1999, O'Really, Sebastopol, CA.

Macy, M. W., R. Willer (2002). *From Factors to Actors: Computational Sociology and Agent-Based Modeling*. Annual Review of Sociology, Vol. 28(1). pp. 143-166.

Mason (2004). *Mason: Multiagent Simulation Toolkit*. 2004. Available from: http://cs.gmu.edu/~eclab/projects/mason/

Minar, N., R. Burkhart, C. G. Langton, M. Askenazi (1996). *The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations*. In: Santa Fe Institute Working Paper: 96-06-042. 1996.

Moretti, S. (2002). *Computer Simulation in Sociology: What Contribution?* Social Science Computer Review, Vol. 20(1). pp. 43-57.

Moss, S. (1998). *Social Simulation Models and Reality: Three Approaches*. In: J. Sichman et al (Eds.): Multi-Agent-Based Simulation, Second International Workshop, MABS 1998, Lecture Notes in Computer Science, Vol. 1534. Springer Berlin-Heidelberg, Paris, France, 1998, pp. 45-60.

Moss, S., B. Edmonds, S. Wallis (1997). *Validation and Verification of Computational Models with Multiple Cognitive Agents*. CPM Report, 1997.

Moss, S., H. Gaylard, S. Wallis, B. Edmonds (1998). *SDML: A Multi-Agent Language for Organizational Modelling*. Computational and Mathematical Organization Theory, Vol. 4. pp. 43-70.

Nagel, E. (1961). The structure of science; problems in the logic of scientific explanation. 1961, Harcourt, Brace and World, New York.

NetLogo (2004). *NetLogo: Center for Connected Learning and Computer-Based Modeling*. 2004. Available from: http://ccl.northwestern.edu/netlogo/

Newell, A. (1982). *The Knowledge Level*. Artificial Intelligence, Vol. 18. pp. 87-127.

Nicolis, G., I. Prigogine (1977). Self-organization in nonequilibrium systems: from dissipative structures to order through fluctuations. 1977, Wiley, New York.

Nicolis, G., I. Prigogine (1989). Exploring Complexity. 1989, Freeman and Co., New York.

OMG (2003a). *MDA Guide, v1.0.1, Object Management Group*. omg/03-06-01, June, 2003a. Available from: http://www.omg.org/cgi-bin/doc?omg/03-06-01

OMG (2003b). *Meta Object Facility (MOF) Specification Version 2.0, Object Management Group*. Core Final Adopted Specification, ptc/03-10-04, October, 2003b.

OMG (2003c). *Unified Modeling Language Specification (UML) Version 2.0, Object Management Group*. UML 2.0 Infrastructure Final Adopted Specification, ptc/03-09-15, November, 2003c. Available from: http://www.omg.org

OMG (2007). *MOF QVT Final Adopted Specification*. ptc/05-01-01, March, 2007. Available from: http://www.omg.org/docs/ptc/05-11-01.pdf

Ostrom, T. M. (1988). *Computer simulation: The third symbol system*. Journal of Experimental Social Psychology, Vol. 24(5). pp. 381.

Parunak, H. V. D., S. John, C. Steve (1998). *Toward the Specification and Design of Industrial Synthetic Ecosystems*. In: Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages. 1998, Springer-Verlag.

Pascual, J. A. (2006). *Modelado Multiagente de Mercados Financieros: Un Enfoque Basado en el Comportamiento Individual de los Inversores*. In: A. López Paredes and J. Pajares Gutiérrez (Advisors): Escuela Técnica Superior de Ingenieros Industriales. 2006, Universidad de Valladolid.

Pavón, J., M. Arroyo, S. Hassan, C. Sansores (2006). *Simulación de Sistemas Sociales con Agentes Software*. Actas del Campus Multidisciplinar en Percepción e Inteligencia, CMPI 2006, Vol. I. Albacete, Spain, July 10-14, 2006, pp. 389-400.

Pavón, J., M. Arroyo, S. Hassan, C. Sansores (2007a). *Agent Based Modelling and Simulation for the Analysis of Social Patterns*. In: Pattern Recognition Letters, Special Issue on Pattern Recognition in Interdisciplinary Perception and Intelligence, Elsevier Science, 2007a.

Pavón, J., J. Gómez-Sanz (2003). *Agent Oriented Software Engineering with INGENIAS*. In: V. Marík et al (Eds.): Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, LNCS, Vol. 2691. Springer, Prague, Czech Republic, June 16-18, 2003, pp. 394-403.

Pavón, J., J. Gómez-Sanz, R. Fuentes (2002). *The INGENIAS Methodology and Tools*. In: Agent-Oriented Methodologies, Idea Group Publishing, 2002, pp. 236-276.

Pavón, J., C. Sansores (2006). *Using INGENIAS for Visual Modeling of Complex Agent Based Simulation Systems*. In: Agent-Based Modelling in Natural Resource Management, edited by A. López-Paredes and C. Hernández, Pearson Education, Madrid, 2006, pp. 173-207.

Pavón, J., C. Sansores, J. Gómez-Sanz (eds.) (2007b). Modelling and Simulation of Social Systems with INGENIAS. International Journal of Agent Oriented Software Engineering, 2007b, Inderscience Publishers.

Peckham, J., F. Maryanski (1988). *Semantic Data Models*. ACM Computing Surveys, ISSN 0360-0300, Vol. 20(3). pp. 153-189.

Phillips, D. C. (1976). Holistic thought in social science. 1976, Stanford University Press, Stanford, CA.

Prigogine, I. (1980). From Being to Becoming: Time and complexity in the physical sciences. 1980, W. H. Freeman, San Francisco.

Ramanath, A. M., G. N. Gilbert (2003). *Towards a Methodology for Agent Based Social Simulation Research*, 2003.

Rand, W., D. G. Brown, S. E. Page, R. L. Riolo, L. E. Fernandez, M. Zellner (2003). *Statistical Validation of Spatial Patterns in Agent-Based Models*. Proceedings of Agent Based Simulation, Montpellier, France, 2003.

Repast (2004). *Repast: Recursive Porus Agent Simulation Toolkit*. 2004. Available from: http://repast.sourceforge.net

Ritzer, G. (2000). Modern sociological theory. 5th ed. 2000, McGraw-Hill, Boston.

Sansores, C., J. Pavón (2004a). *Agents on Social Sciences: ABSS Case Study*. Proceedings of the Fifth Iberoamerican Workshop on Multi-agent Systems (IBERAGENTS 2004), Puebla, México, November 23, 2004a.

Sansores, C., J. Pavón (2004b). *A Framework for Agent Based Social Simulation*. Proceedings of the Second European Workshop on Multi-Agent Systems (EUMAS 2004), Barcelona, Spain, December 16-17, 2004b.

Sansores, C., J. Pavón (2004c). *MAS Scalability: ABSS on the Grid Case Study*. Proceedings of the Third International Workshop on Practical Applications of Agents and Multi-agent Systems (IWPAAMS 2004), Burgos, Spain, October 13-15, 2004c.

Sansores, C., J. Pavón (2005a). *Agent-Based Simulation Replication: a Model Driven Architecture Approach*. In: A. Gelbukh et al (Eds.): Fourth Mexican International Conference on Artificial Intelligence (MICAI-2005), Lecture Notes in Artificial Intelligence, Vol. 3789. Springer-Verlag, Monterrey, México, November 14-18, 2005a, pp. 244-256.

Sansores, C., J. Pavón (2005b). *Agent Based Modeling of Social Complex System (Thesis Proposal)*. In: R. Marín et al (Eds.): 11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2005), Revised and Invited Papers (2006), Current Topics in Artificial Intelligence, Lecture Notes in Artificial Intelligence, Vol. 4177. Springer-Verlag, Santiago de Compostela, Spain, November 14-15, 2005b, pp. 99-102.

Sansores, C., J. Pavón (2005c). *Agent Based Simulation for Social Systems: from Modeling to Implementation*. In: R. Marín et al (Eds.): 11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2005), Revised and Invited Papers (2006), Current Topics in Artificial Intelligence, Lecture Notes in Artificial Intelligence, Vol. 4177. Springer-Verlag, Santiago de Compostela, Spain, November 14-15, 2005c, pp. 79-88.

Sansores, C., J. Pavón (2005d). *Simulación Social Basada en Agentes*. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial "Aplicaciones de la Tecnología de Agentes", ISSN 1137-3601, Vol. 9(25). pp. 71-78.

Sansores, C., J. Pavón (2006). *Agent Based Simulation for Social Systems: from Modeling to Implementation*. In: R. Marín et al (Eds.): 11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA'05), Current Topics in Artificial Intelligence, LNAI, Vol. 4177. Springer-Verlag, Santiago de Compostela, Spain, 2005, 2006, pp. 79-88.

Sansores, C., J. Pavón, J. Gómez-Sanz (2005). *Visual Modeling for Complex Agent-Based Simulation Systems*. In: J. Sichman and L. Antunes (Eds.): Multi-Agent-Based Simulation, Sixth International Workshop on Multi-Agent-Based Simulation (MABS 2005), Revised and Invited Papers (2006), Lecture Notes in Computer Science, Vol. 3891. Springer-Verlag, Utrech, Netherlands, July 25, 2005, pp. 174-189.

Sansores, C., J. Pavón, A. López-Paredes (2004). *Towards a Framework for ABSS on the Grid*. Proceedings of the Second Conference of the European Social Simulation Association (ESSA 2004), ISBN 84-688-7964-9, Valladolid, Spain, September 16-19, 2004.

Sawyer, R. K. (2001). *Emergence in Sociology: Contemporary Philosophy of Mind and Some Implications for Sociological Theory*. American Journal of Sociology, Vol. 107(3). pp. 551-585.

Sawyer, R. K. (2005). Social Emergence: Societies as Complex Systems. 2005, Cambridge University Press, Cambridge.

Schelling, T. C. (1971). *Dynamic Models of Segregation*. Journal of Mathematical Sociology, Vol. 1. pp. 143-186.

Schelling, T. C. (1978). Micromotives and Macrobehavior. 1978, Norton, New York.

Schmidt, D. C. (2006). *Model Driven Engineering*. IEEE Computer, Vol. 39(2). pp. 25-31.

Schreiber, D. J. D. (2002). *Validating Agent-Based Models: From Metaphysics to Applications*. Proceedings of Midwestern Political Science Association's Annual Conference, Chicago, USA, 2002.

SDG (2006). *Swarm Development Group*. 2006. Available from: http://www.swarm.org

SDML (1997). *SDML: a Strictly Declarative Modelling Language*. 1997. Available from: http://www.cpm.mmu.ac.uk/sdml

SeSAm (2004). *SeSAm: Integrated Environment for Multi-Agent Simulation*. 2004. Available from: http://www.simsesam.de/

Shoham, Y. (1993). *Agent Oriented Programming*. Artificial Intelligence, Vol. 60(1). pp. 51-92.

Simon, H. A. (1962). *The Architecture of Complexity*. Proceedings of the American Philosophical Society, Vol. 106(6). pp. 467-482.

Simon, H. A. (1982). Models of bounded rationality. 1982, MIT Press, Cambridge, MA.

Simon, H. A. (1996). The sciences of the artificial. 3rd ed. 1996, MIT Press, Cambridge, Mass.

StarLogo (2004). *StarLogo: on the web*. 2004. Available from: http://education.mit.edu/starlogo/

Steels, L. (1995). *Building Agents out of Autonomous Behavior Systems*. In: The Artificial Life Route to Artificial Intelligence, edited by L. Steels and R. Brooks, Lawrence Arlbaum Associates, Inc, New Jersey, 1995.

Sun, R. (ed.) (2005). Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation, 2005, Cambridge, Cambridge University Press, p. 448.

Swarm (2004). *Swarm Wiki: the agent-based modelling resource*. 2004. Available from: http://wiki.swarm.org

Sykara, K. (1998). *Multi-Agent Systems*. AI Magazine, Vol. 19(2). pp. 85-92.

Tisue, S., U. Wilensky (2004). *NetLogo: Design and Implementation of a Multi-Agent Modeling Environment*. SwarmFestSwarm Development Group, Ann Arbor, MI, 2004.

Tobias, R., C. Hofmann (2004). *Evaluation of free Java-libraries for Social-Scientific Agent Based Simulation*. Journal of Artificial Societies and Social Simulation, Vol. 7(1).

Tolvaner, J.-P. (2000). GOPRR Metamodelling Languaje. 2000.

Troitzsch, K. G. (1997). *Social Simulation -- Origins, Prospects, Purposes*. In: R. Conte et al (Eds.): Simulating Social Phenomena, Vol. 456. Springer-Verlag, Berlin, Heidelberg, New York, Berlin, 1997, pp. 41-54.

Varela, F. J. (1979). Principles of biological autonomy. 1979, North Holland, New York.

Von Foerster, H. M. (1981). Observing Systems: Selected papers of Heinz von Foerster. 1981, Intersystems, Seaside, CA.

Von Foerster, H. M. (1996). Cybernetics of Cybernetics. 2nd ed. 1996, Future Systems, Minneapolis.

Von Neumann, J., O. Morgenstern (1953). Theory of games and economic behavior. 3d ed. 1953, Princeton University Press, Princeton.

Waldrop, M. M. (1992). Complexity: The Emerging Science at the Edge of Order and Chaos. 1992, Simon & Schuster, New York.

Wasserman, S., K. Faust (1994). Social network analysis: methods and applications. In Structural analysis in the social sciences; 8. 1994, Cambridge University Press, Cambridge; New York.

Watkins, J. W. N. (1955). *Methodological Individualism: A reply*. Philosophy of Science, Vol. 22. pp. 58-62.

Wilensky, U. (1999). *Center for Connected Learning and Computer-Based Modeling, Northwestern University*. Evanston, IL. 1999. Available from: http://ccl.northwestern.edu/netlogo/

Wilensky, U., W. Stroup (1999). *HubNet*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, IL. 1999.

Wilkinson, G. S. (1984). *Reciprocal food sharing in the vampire bat*. Nature, Vol. 308(5955). pp. 181-184.

XML (2006). *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C Recomendation, October, 2006. Available from: http://www.w3.org/TR/2006/REC-xml-20060816

Zeleny, M. (ed.) (1981). Autopoiesis: A Theory of Living Organization. North Holland Series in General Systems Research, Vol. 3, 1981, New York, North Holland.

Glosario

ABSS	*Agent Based Social Simulation*
AI	*Artificial Intelligence*
AS	*Artificial Societies*
ASML	*Artificial Societies Modeling Language*
API	*Application Programming Interface*
BDI	*Beliefs-Desires-Intentions*
CIM	*Computational Independent Model*
DAI	*Distributed Artificial Intelligence*
GOPRR	*Graph, Object, Property, Relationship, and Role*
LMSA	*Lenguaje de Modelado de Sociedades Artificiales*
IDK	*INGENIAS Development Kit*
ISM	*Implementation Specific Model*
MAS	*Multi-Agent System*
MDE	*Model Driven Engineering*
MOF	*Meta Objects Facilities*
OMG	*Object Management Group*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
SA	*Sociedad Artificial*
SMA	*Sistema Multi-Agente*
SSBA	*Simulación Social Basada en Agentes*
UML	*Unified Modeling Language*
XML	*eXtended Modelling Language*