

Requirements Analysis of Agent-Based Simulation Platforms: State of the Art and New Prospects

Maria Bruno Marietto^{1,†}, Nuno David^{1,2,‡},
Jaime Simão Sichman^{1,¥} and Helder Coelho³

¹ Intelligent Techniques Laboratory, University of São Paulo, Brazil
gracas.marietto@poli.usp.br <http://www.lti.pcs.usp.br/~graca>
jaime.sichman@poli.usp.br <http://www.pcs.usp.br/~jaime>

² Department of Information Science and Technology, ISCTE/DCTI, Lisbon, Portugal
Nuno.David@iscte.pt <http://www.iscte.pt/~nmcd>

³ Department of Informatics, University of Lisbon, Portugal
hcoelho@di.fc.ul.pt <http://www.di.fc.ul.pt/~hcoelho>

Abstract. In this paper we propose a preliminary reference model for the requirements specification of agent-based simulation platforms. We give the following contributions: (i) aid the identification of general principles to develop platforms; (ii) advance the analysis and prospection of technical-operational and high-level requirements; (iii) promote the identification of shared requirements, addressing them to the development of an integrated work. We present our reference model and make a comparative analysis between three well-known platforms, resulting in an unambiguous and schematic characterisation of computational systems for agent-based simulation.

1 Introduction

Computational platforms and test-beds are becoming increasingly important in Multi-agent Systems (MAS). A first goal of these systems is to release the researchers from low-level technical-operational issues, allowing the researcher to concentrate his/her efforts on the relevant domain application logic. In the area of Multiagent-Based Simulation (MABS) (see [22]), computational platforms are important methodological tools that aid the researcher in the processes of modelling and development of simulation programs. They are especially valuable to conciliate different interdisciplinary perspectives, which in MABS typically involve researchers from various scientific areas, such as social psychology, artificial intelligence, computer engineering, artificial life, social biology, sociology and economics. The interdisciplinary

[†] Supported by FAPESP, Brazil, grant number 00/14689-0. On leave from Dom Bosco Catholic University.

[‡] Partially supported by FCT/PRAXIS XXI, Portugal, grant number BD/21595/99.

[¥] Partially supported by CNPq, Brazil, grant number 301041/95-4, and by project MAPPEL (PROTEM-CC, CNPq/NSF), grant number 680033/99-8.

character of MABS is an important challenge faced by all researchers in this discipline, while demanding a difficult interlacement of different methodologies, terminologies and points of view. To help with this process of integration, simulation platforms that support educational, industrial and scientific research objectives are in increasing demand.

Meanwhile, at the present maturing stage of the discipline, the *requirements analysis* of simulation platforms plays a fundamental role, since there is not yet a consensus with respect to the specification of services and its standards of behaviour. It is important to include this subject in the MABS agenda because:

1. The process of requirements analysis promotes a deeper discussion of many research topics in the area, such as the problem of observation of emergent phenomena, verification and validation of models. If such topics and associated solutions are not well defined than that will become evident during the design of both the general structure and the global behaviour of platforms, helping the researchers to redirect their works. Conversely, if they are already well defined, requirements analysis will bring new dimensions to accepted solutions and further integration with other requirements.
2. The process of requirements analysis identifies shared requirements of many projects, addressing them to the development of an integrated work rather than individual ones.

In effect, when evaluating the importance of requirements analysis in this field it is quite odd to find very few references in the literature about this topic. This observation becomes even more surprising since one can find a considerable number of platforms (though very heterogeneous) available to the research community (see e.g. [3, 12, 16, 19, 20, 25]). The extensive availability of domain specific and general-purpose platforms does not necessarily facilitate the work of researchers in the field. The diversity of functions and diffusion of ends can bring benefits, but also disadvantages in the absence of reference models that help the researcher to systematize his/her choices and needs. Presently, a clear and systematised reference model is in need, in order to stimulate the integration of different works and materialize new prospects for requirements related to common problems in the field, such as the observation and manipulation of emergent phenomena.

Aiming to assist on the construction of a general framework that characterizes an ideal type of platform, the *SimCog* project [23] aims at two independent and cross-fertilizable goals. The first goal is to define a *reference model* for the requirements specification of an ideal type MABS platform. The second goal is the specification, design and implementation of a platform complying with a subset of those requirements, with special focus on the simulation of cognitive agents. With regard to the first goal a set of functional and non-functional requirements guiding the specification of MABS platforms may be found in [17]. Such specification will be further validated and specialized based on two independent approaches: (i) by making a comparative analysis between requirements of different platforms that are presently available to

the research community; (ii) by prospecting and exploring requirements with researchers in the field¹. In this work we report our results with respect to the first approach, the comparative analysis.

This paper is structured as follows. In section 2 we will present a set of requirements that seem to characterize both the general structure and global behaviour of MABS platforms. In section 3 we will evaluate the adequacy of the reference model with a comparative analysis between requirements of three well-known platforms (CORMAS, Swarm and MadKit). The comparative analysis will allow us to trace: (i) the set of common requirements that characterize all platforms; (ii) the set of requirements that do not characterize any platform, bringing new dimensions to the prospection of requirements in the *SimCog* reference model. Finally, in section 4, we draw some conclusions and suggest some general principles that should be considered in the design of MABS platforms.

2 Requirements Analysis of MABS Platforms

Currently, there are a large number of computational systems in MABS. While analysing such systems it is possible to detect several technologies, but among this diversity there are certain groups of requirements that characterize different kinds of technologies. Such groups of requirements will be called *facilities*, borrowing the term from [9]. We identify four facilities that can be found in these computational systems: *technological*, *domain*, *development* and *analysis*. In this work, computational systems that present at some degree of development these four facilities, will be called *MABS platforms*.

Meanwhile, there are a number of requirements that are not so well systematized and developed. Most of them focus on the exploration of unexpected outputs or emergent structures that should desirably play causal effects in the evolution of simulation results. These requirements are important to balance the subjective role of observing and validating unexpected results with the objective role of verifying such results. We will cluster these services in a new group called *exploration facilities*.

In [17] we present a requirements specification for MABS platforms. As stated by [21], a requirement is a feature of a system or a description of something the system is capable of doing in order to reach its objectives. Additionally, it aims to detail the structure of a system, establishing its principles of behaviour. The requirements presented in [17] followed a top-down approach, usually adopted in requirements engineering. At first the most general requirements are described and at subsequent levels they are made more specific. In that specification there is a description of forty seven (47) requirements, distributed across the five facilities. The requirements were classified in two categories, typical in requirements engineering: functional and non-

¹The survey with researchers in the field can be found at http://www.lti.pcs.usp.br/SimCog/Doc/M1_Questionnaire.html. This survey has involved more than a hundred and seventy opinions of recognized researchers in MABS. Queries will be available at http://www.lti.pcs.usp.br/SimCog/Doc/Search/M1_Quest.html.

functional. Functional requirements describe the behaviour and the services that the user expects a system to provide. Non-functional requirements describe restrictions, qualities and/or benefits associated with the system, but usually do not specify a detailed account of services in the system.

In this section we present a small subset of these requirements. We restrict our presentation to functional requirements. The selection was guided with the aim of identifying those that (i) seem to compose a basic structure of MABS platforms; (ii) are common to, or that better differentiate, current existing platforms, and (iii) are not present in any platform that we know, at least in a formal way. In this presentation the following points are considered:

1. A brief description of each requirement and some associated activities in both a precise and unambiguous way. This approach is important in MABS due to its interdisciplinary character; for instance the term "domain" must be interpreted in the same way by a Psychologist, an Anthropologist, an Engineer, and so on;
2. Each functional requirement has a name. In order to connect requirements with UML use cases [2], each name is an active phrase, for instance "Launch Agents".

In the rest of this section we will describe requirements clustered around the fore-said facilities. In section 3 we will use these requirements to guide the comparative analysis.

2.1 Technological Facilities

Technological facilities encompass services that (i) intermediate the platform with both the operational system and the network services; (ii) provide services to support controlled simulation worlds.

A. Manage Scheduling Techniques: The platform should support controlled simulations and allow repeatability. To this end it should provide (i) libraries including at least one commonly used scheduling technique, like discrete-time or event-based simulation; (ii) mechanisms to cluster agents in groups and apply different scheduling techniques to each group.

2.2 Domain Facilities

According to [1], the environment of a problem can be represented by a collection of objects and agents. In our sense, a domain is defined by the environment and causal knowledge of how those objects and agents interact with one another. Domain facilities embrace two sub-types of requirements:

- The first type deals with requirements that have a considerable importance in the modelling and implementation of domains. This importance is assessed by consid-

ering how the absence of requirements may hamper, or even inhibit, the modelling and implementation of domains. This is the case of §2.2.A and §2.2.C.

- The second type deals with requirements whose technological and logical functionalities must be modelled in a personalized way according to the relevant domain. They usually demand further implementation work. This is the case of §2.2.B.

A. Launch Agents: The platform should provide agent templates related with different manners to launch agents like threads, applets and objects. For instance, platforms that do not provide multi-threaded agents may hamper the modelling and simulation of distributed network environments.

B. Manage Intentional Failures: From a technical-operational point of view, there are two classes of intentional failures that can be manipulated in a simulation (see [11]). The first class, called *operational failures*, works with disturbances in the technical-operational infrastructure (corrupted messages, server failures, etc.). The second, called *logical failures*, manipulate patterns of behaviour that can be viewed as dysfunctional exceptions in the simulated system. Operational failures can be used to build specific scenarios and serve as the base to build more general logical failures. Logical failures are strongly domain dependent, and the user may have to engage in further implementation work in order to utilize them. The platform should offer: (i) libraries to manipulate basic operational failures; (ii) mechanisms to store and search templates of logical failures created by users.

C. Integrate Controlled and Non-Controlled Environments: Typically the simulation environment must be totally controlled, every event in the simulation world must be performed under the control of the simulator. These situations characterize what we call *controlled environments*. Nevertheless, there are cases where the agents can (or must) perform actions outside the controlled environment, in real environments. This integration can occur in two ways. In a first scenario an agent could use the platform environment to perform some of its actions, while performing others under real conditions. In a second scenario, an agent could be decoupled from the simulator, perform autonomously some action under real conditions, and return to the simulated environment. In both cases this integration demands a time-consuming implementation work from the developer.

To support this functionality a platform should offer agent architectures that separate the agent domain-dependent behaviour from the simulator design patterns. Also, in order to keep the simulation consistent and guarantee a good level of repeatability, when agents are running in non-controlled environments some of their events should be notified to the simulator, which should update its local view (see [25]).

2.3 Development Facilities

Development facilities include mechanisms and tools to construct MAS within an agent-centered approach or organisation-centered approach (see [15]).

A. Develop Agent Architectures: The platform should provide templates of generic agent architectures, from reactive to intentional agents.

B. Manage Messages: Agents should communicate between each other with message passing mechanisms. The platform should also offer: (i) message models with basic attributes and mechanisms to extend or add additional attributes; (ii) APIs and parsers to check the correctness of agent communication languages, eventually according to given standards, like *e.g.* KQML.

C. Use Organisational Abstractions: Organisational abstractions are MAS components that explicitly structure an organisation. Roles and groups are the most common ones. The platform should provide services that define roles in terms of, *e.g.*, functions, activities or responsibilities. It should also be possible to define their interaction protocols (see [26]). The concept of groups is an example of an organisational abstraction that defines different sub-organisations according to modularity and encapsulation principles. The platform should support the creation and management of agent and organisation collections, clustered around common relations.

D. Use Multiple Societies: In the real world we have the ability to create explicit organisational structures, observe them and reason about them, like other agents, institutions or even new societies (for example, artificial agent societies). From the observer's point of view, an artificial society may be seen as an aggregate of agents and organisations, which coexist and interact with each other through social events. This concept of society in agent-based simulation is rarely specified as an explicit structural and relational entity, but usually implicitly defined in formal or informal terms of inclusiveness of agents and other organisational entities [7]. Such tendency complicates the design of artificial agents that are able to observe and reason about other societies, particularly if the environment is composed of multiple interacting social spaces and levels of abstraction. Although some approaches have used models that explicitly define multiple societies, the concept of society in those models is still reducible to the concept of group, where agents are viewed simultaneously both as actors and non-neutral observers in a given society. Therefore the role of opaque artificial observation is not explicitly assigned to agents, being exclusively and implicitly defined in the person of the system designer

The possibility of instantiating multiple societies in an explicit way, as explicit organisational and relational entities themselves, can be very useful in MABS. It is important, for instance, in multi-level modelling approaches (hierarchical organisations of social spaces). Additionally, it can serve as the basic mechanism to integrate controlled and non-controlled environments (see §2.2C). The platform should provide primitives to instantiate topologies of multiple societies, and to instantiate opaque social spaces (see §2.5C, [7]) that may be used as neutral observation windows to other societies and social spaces.

E. Use Ontologies: Ontologies can be used with many objectives like interoperability, reuse, search of information and knowledge sharing and acquisition (see [24, 13]). Ontologies can be powerful tools to assist simulation modelling. The platform should provide: (i) a standard ontology describing its adopted concepts and their causal relationships; (ii) mechanisms to store new ontologies based on the standard ontology; (iii) a search engine to manipulate ontologies; (iv) a browser to query the search engine and visualize results; and (v) mechanisms to relate ontologies with implementation components.

2.4 Analysis Facilities

The researcher's capacity to observe and interpret simulation results can lead someone to ask if those results are verified, validated and relevant to the observed system, questioning if such results are only conclusions associated with a specific observer. This kind of question is found in [4], where the author points out the need to construct structured simulations in order to explain the emergence and "imergence" phenomena.

According to [4], there is not yet a "...real emergence of some causal property (a new complexity level of organisation of the domain); but simply some subjective and unreliable global interpretation". In [8] we show that this problem arises in MABS because the *computational model* may not be complete, or even specified correctly, in relation to the intended *conceptual model* specification – see figure 1. Additionally, the *conceptual model* may not be complete, or even specified correctly, in relation to the observed target system. In fact, even if the observer is able to observe certain patterns in the simulation results that could, according to some possible world, affect the intended conceptual model, the underlying observation (e.g., a law, a theorem) may have not been specified in the computational model, bringing the observed result to the realm of epiphenomenalism with no causal power on the computational model.

Nevertheless, it is possible to specify requirements that can help the researcher to observe objective simulation results. In [8] we view agent-based computational models as a triple composed by (i) a set of *Agentified Entities* (AE); (ii) a set of properties observed over each one of those AEs; and (iii) a set of interaction algorithms. Agentified entities may be (i) atomic entities representing an observed feature, agent or organisation in the target; or (ii) aggregates specified with interaction algorithms that manipulate atomic or other aggregate entities according to properties observed over those entities. The former are called *micro-AEs* and the latter are called *macro-AEs*. Macro AEs may differ in order of level according to different aggregation levels. The AEs interact with each other through primitives that define the set of behavioural events specified in the model (e.g. message-passing, perception mechanics). Requirements related to *analysis* should specify the means to observe behavioural events and the internal state of AEs during the simulation. Furthermore they should specify the means to define windows of observation that can be cross analysed, like the effects produced by some micro-AE's behavioural event in the internal state of some macro-AE (and *vice versa*).

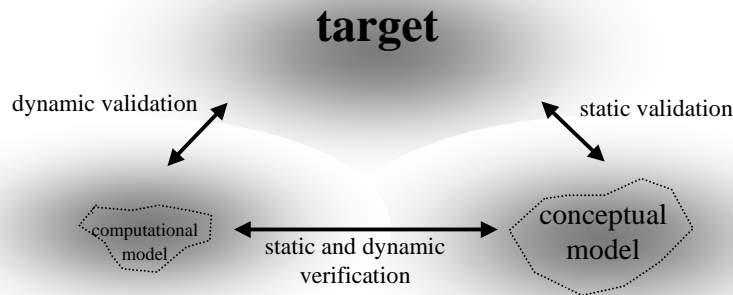


Figure 1 - Conceptual and computational models. Static verification and validation techniques are concerned with the analysis and checking of system representations such as specification documents and the program source code. Dynamic testing exercises the program using (real or experimental) data, and the values obtained are used to test if the computer program behaves as expected by the conceptual model (verification) or as observed in the target (validation) – see [8].

Figure 2 acknowledges the information processing character of computational models, and illustrates lines of information flow which can be observed at different levels of aggregation. The figure emphasizes the effect of macro-AEs on micro-AEs (or lower order macro AEs), and *vice versa*. Continuous lines denote flow of information caused by invocation of behavioural events, where each AE acts as an information transformation function. The symbol “x” denotes observation of behavioural events or internal states of AEs. Dotted lines denote control activity exercised by the simulator.

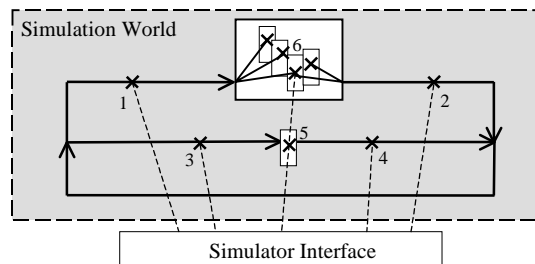


Figure 2 - Points of observation and intervention on behavioural and cognitive events.

A. Observe Behavioural Events: Behavioural events are events that can be observed by an external observer (e.g., message passing, creation/destruction of agents, data base access). The platform should provide mechanisms to select specific windows to observe behavioural events. Observation windows for behavioural events are given in figure 2 by points (1), (2), (3) and (4). Points (1) and (2) observe whole sets of behav-

avioural events issued by AEs associated with a same macro-AE. Points (3) and (4) refer to events associated with a single AE.

B. Observe Cognitive Events: Cognitive events are events in the agents' internal architectures. The observation of such events may be counter-intuitive according to the usual agent paradigm, but indispensable to analyse structured simulations. For example, it may be useful to analyse the effect of some behavioural event in the recipient agent's mental states; or if the effect perceived by the agent who issues the event is in fact objective according to the simulation designer-observer's perspective.

The platform should offer mechanisms to control the agents' internal mechanisms, in order to trigger specific observation methods. In figure 2 point (6) denotes the observation of whole sets of AEs associated with a same macro-AE. Point (5) denotes the observation of single AEs. In order to provide structured observation of cognitive events, the platform should comply with another requirement, called cognitive reflectivity (see §2.5D).

C. Provide Data Analysis: In a simulation the amount of output data is huge and graphical analysis at run time cannot capture all aspects of the simulation logic. This requirement should define technical indicators and decision support (e.g., graphical and statistical packages) to work in more depth with generated data.

2.5 Exploration Facilities

Exploration facilities emphasise the human-computer interactive character of simulations with respect to the exploration of different results and emerging qualitative concepts. This is important because most classic simulation programs and contemporaneous models of agent-oriented software engineering (see [5]) are not concerned with the exploration of emergent, non-anticipated, structures. While most classic software processes concentrate on the analysis and exploration of system requirements and intended behaviours, the MABS software process is also concerned with *exploration of results*. The interactive exploration of different conditions, such as different sequences of method invocation, mental states or assignment of variables, is thus a crucial issue. The exploration can be facilitated if those conditions are allowed to change interactively, during the simulation, in-between simulation steps.

A. Intervene in Behavioural Events: Behavioural events are events that can be observed by an external observer. The platform should offer mechanisms to select specific points to intervene in behavioural events. The intervention should permit the *suppression, modification* or *creation* of behavioural events. For instance, it may be useful to *modify* the content of behavioural events (e.g., the intended recipient agent in a message), or even to *suppress* its arrival to the intended recipient. These experiments give the means to analyse functional effects in the simulation independently from the agents' internal representations that originate other or those same events. In figure 2, points (1) and (2) refer to intervention in sets of behavioural events issued by AEs associated with a same macro-AE. Points (3) and (4) denote intervention in behavioural events issued by a single AE.

B. Intervene in Cognitive Events: The platform should offer mechanisms to intervene in the agents' internal mechanisms, for instance modify the values of program variables, order of method invocation and the agents' beliefs. This may alter the order of invocation and the nature of behavioural events. Point (6) denotes the intervention in AEs associated with a same macro AE, and point (5) in a single AE. In order to provide structured interventions the platform should comply with requirement §2.5D, called cognitive reflectivity.

C. Manage Social Opacity: The problematic of social opacity analyses the organisational conditions under which the control of cognitive information transfer between different social spaces at the same level of abstraction is possible [7], for instance between different multiple societies (see §2.3D). Social opacity is therefore related to organisational borders.

The platform should provide the means to instantiate different topologies of opaque social spaces in a dynamic way. This is useful to simulate agents that have the ability to instantiate and observe given models of other artificial agents and societies, allowing the agents to reason autonomously about the heterogeneity of different models of societies at various levels of observation. Nevertheless, while the observed agents and societies must be visible to the observer agent, the observer agent and societies must be opaque to the observed agents. The platform should provide organisational ingredients and services to instantiate multiple societies and opaque social spaces.

D. Provide Models of Cognitive Reflectivity: Cognitive reflectivity refers to the identification of cognitive structures and internal procedures of agents at run time. The templates of agent architectures (see §2.3A) should provide adequate models of cognitive reflectivity, allowing the user or other system agents to observe and intervene in the simulated agents' cognitive events. Thus, different models of cognitive reflectivity should be provided according to different generic agent architectures.

3 Comparative Analysis of MABS Platforms

We are now in conditions to present the platforms that will be subject to the comparative analysis: CORMAS, Swarm and MadKit. The main purpose of CORMAS and Swarm is to simulate agent-based models of social and biological targets. CORMAS is a domain dependent platform in the area of renewable natural resources. Swarm is a general-purpose platform to simulate complex adaptative systems. MadKit is an organisation-centered platform based on MAS organisations. It is a general-purpose simulation platform, although more oriented to MAS engineering.

The Common-pool Resources and Multi-Agent Systems (CORMAS) platform is being developed by the Centre de Coopération Internationale en Recherche Agronomique pour le Développement (CIRAD), France. The platform supports interactions between individuals and groups sharing renewable natural resources (see [3, 6]). CORMAS works with three types of entities: spatial, passive and social. Spatial entities define a topological support for simulations. They hold natural resources and

arbitrate their allocation according to pre-defined protocols based on a metaphor of physical harvest. Passive entities are subdivided in passive objects and messages; and social entities are agents.

Table 3.1 - Comparative outline of MABS platforms.

	<i>CORMAS</i>	<i>MadKit</i>	<i>Swarm</i>
Technological Facilities			
Manage Scheduling Techniques	Adopts discrete time simulation	The couple Scheduler-Activator define and manage personalized scheduling techniques	Adopts event-based simulation
Domain Facilities			
Launch Agents	Agents can be launched as objects	Agents can be launched as objects, threads or applets	Agents can be launched as objects or threads
Manage Intentional Failures	Not available	Not available	Not available
Integrate Controlled and Non-Controlled Environments	Not available	Not available	Not available
Development Facilities			
Develop Agent Architectures	Flat architecture	Flat architecture	Flat architecture
Manage Messages	Synchronous and asynchronous modes	Synchronous and asynchronous modes	Synchronous mode
Use Organisational Abstractions	Does not work with roles. It works with agents aggregations, through the Group class	There are many-to-many relation among agents and roles. Each group has its inner characteristics	Does not work with roles and groups
Use Multiple Societies	Not available	Groups do not quite resemble the concept of societies, but can be made alike	Multiple societies are swarms, but must be allocated in hierarchical levels
Use Ontologies	Not available	Not available	Not available
Analysis Facilities			
Observe Behavioural Events	Not formally available	Hook mechanisms and system agents	Not formally available
Observe Cognitive Events	Designer implements methods in SmallTalk	Watcher agents and Probe class	Observer agents and probe interface
Provide Data Analysis	Provides a rich set of tools	Not available	Libraries like sim-tools and random

The Multi-Agent Development Kit (MadKit) platform is being developed by Jacques Ferber and Olivier Gutknecht of the Laboratoire d'Informatique, Robotique et Micro-Electronique of Montpellier (LIRMM), France. MadKit is based on the 3-tupla

Agent-Group-Role (AGR). The platform adopts the principle of agentification of services where all services are modelled using the AGR concepts, except for the services provided by the micro-kernel. Services are defined through roles and delegated to agents allocated in groups (see [10, 16]).

In the Santa Fe Institute, U.S.A., Chris Langton initiated the Swarm project in 1994. Swarm is a multi-agent platform to simulate complex adaptive systems. The main component that organizes agents is a *swarm*, a collection of agents with a schedule of agent activities. Agents interact with each other through discrete events (see [14,19]).

In Table 3.1 we present a comparative overview of CORMAS, MadKit and Swarm. Exploration facilities were not considered since none of the platforms meets those requirements, at least in a formal and structured way. We will present some general principles to guide the development of exploration facilities in section 4.

3.1 Technological Facilities

In general, we found that requirements related with technological facilities are well structured in all platforms.

A. Manage Scheduling Techniques: CORMAS works with discrete time simulation. As stated by [3] this technique was chosen due to its simplicity. Swarm works with event-based simulation. A swarm is composed by a collection of agents with a schedule of events over those agents. Each event is an action that an agent can perform (see [14,19]). MadKit does not define *a priori* any type of scheduling technique. There are two classes related with scheduling in MadKit. The class `Activator` defines a basic scheduling policy and relates this policy to a set of agents. According to [18], the specialization of the class `Activator` permits the customisation of different scheduling techniques for different sets of agents. The class `Scheduler` defines the agents who are responsible for the integration and consistency of these different techniques.

3.2 Domain Facilities

The platforms do not provide a very rich set of domain facilities. This is not very strange since all the platforms are relatively recent. One can expect an improvement on this class of requirements with the use and development of new releases.

A. Launch Agents: Depending on the simulation domain it may be adequate to launch agents as a thread, or as an applet, or as an object. This requirement, determines, for instance, if the platform allows distributed simulation (or easy evolution towards that goal). CORMAS uses a single thread of control in a simulation and its agents are SmallTalk objects. This feature hampers distribution of agents. In MadKit agents can be launched as objects, threads or applets. In Swarm agents can be instantiated as objects in their own threads, allowing the modelling of distributed simulations.

B. Manage Intentional Failures: This requirement is not available in any platform.

C. Integrate Controlled and Non-Controlled Environments: This requirement is not available in any platform.

3.3 Development Facilities

Requirements related to development facilities are well structured in MadKit, since the platform is based on the Aalaadim organisational model [10]. Swarm also adopted an organisation-centered approach, well adapted to model biological systems, but in essence Swarm serves the purpose of agent-centered models. In Swarm, agents can be composed of other swarms in a nested composition, generating organisational structures in hierarchical levels of access and control. CORMAS adopted an agent-centered approach, and its services attend the needs of such modelling.

A. Develop Agent Architectures: CORMAS does not impose restrictions on the agents' internal architectures and the user is fully responsible for such implementation. In MadKit agents are defined as active communicating entities that play roles within groups. Although MadKit does not define *a priori* the agents' internal architectures, it provides templates to help the modelling of reactive and cognitive agents. An agent in Swarm is an entity that is able to generate events that affect other agents and himself. Swarm does not define *a priori* the agents' internal architectures, but provides libraries of agent templates. For instance, the `ga` and `neuro` libraries provide models for genetic algorithms and neural networks, respectively.

B. Manage Messages: CORMAS works with message passing and the user can define their own communication language. The class `Msg` defines a basic syntax with the attributes: `sender`, `receiver` and `symbol` [6]. Distributed communication is not supported in the current version of CORMAS. Both MadKit and Swarm allow local and distributed communication, but in different levels of functionality. Communication in MadKit occurs through asynchronous message passing. The users can define their own agent communication language, and the platform provides various codifications, such as `StringMessage`, `XMLMessage` and `ActMessage`. A MadKit micro kernel handles local message passing between agents. The exchange of messages between different micro-kernels is also possible. In Swarm, agent communication is always synchronous and bounded inside each swarm. Different swarms can run in different machines and processors, but agents located in different swarms cannot communicate with each other. Like CORMAS and MadKit, the user can define their own communication language.

C. Use Organisational Abstractions: CORMAS does not adopt the concept of role. This is understandable since its domain-dependent character does not emphasise the work with cognitive agents and organisation-centered approaches. This also applies to Swarm, since the platform was initially developed according to artificial life principles. A role in MadKit is an abstract representation of an agent function, service or identification within a group. Several agents can play a same role, and a same agent

can play several roles. In CORMAS the concept of group represents a composite social entity; entities in different groups can interact with entities in other groups, so as to simulate interactions between different scales and organisational levels. The concept of group is implemented through the class `Group` and its subclasses (see [6]). In MadKit groups are atomic sets of agent aggregation. Each group has its own inner characteristics, like communication language, ontology, coordination techniques [16]. In Swarm the concept of group is not explicitly defined. The underlying concept of swarm, as a collection of agents, does not quite resemble the concept of group, since swarms in different hierarchical levels are opaque to each other.

D. Use Multiple Societies: The platform CORMAS runs with a single kernel and manages a single society. Considering that communication between different simulations in different kernels is not possible, CORMAS does not work with multiple societies. Conversely, MadKit and Swarm can deal with multiple societies, although in a limited way. In Swarm, each swarm may be viewed as a single society, but the concept of swarm cannot be decoupled from the concept of agent. It is possible to define different swarms allocated in hierarchical levels. However, swarms in higher hierarchical levels are necessarily opaque to swarms in lower levels. A parent swarm can launch and observe agents in swarms of lower levels, and swarms in the same level interact implicitly if they share a same parent swarm. However, a same agent cannot move between different swarms. Since swarms in the same nested level cannot explicitly interact, topologies of multiple societies are necessarily hierarchical. In MadKit the concept of group does not quite resemble the concept of society. This is because the concept of group is created with specific inner organisational characteristics, like the roles an agent is allowed to play or communication links between agents. It is possible to build topologies of multiple societies with groups, but the platform does not provide explicit mechanisms to control levels of observation and visibility between multiple societies (see §2.5.C).

E. Manage Ontologies: This requirement is not available in any platform.

3.4 Analysis Facilities

In CORMAS and Swarm the separation between observation of behavioural and cognitive events is not formally structured, there is not a formal distinction between what is objectively observable in the external environment and what is subjectively represented in the agents' mind. In MadKit this separation is not entirely structured, but seems to be more flexible than in CORMAS and Swarm, partly due to its organisation-centered approach.

A. Observe Behavioural Events: In CORMAS the designer needs to implement methods in SmallTalk to operate the observation of behavioural events, and provides sophisticated tools to observe such events. Swarm decouples the observation actions from execution actions, working with a two level hierarchical architecture. In the first level occurs the observation of the simulation (the Observer Swarm). In the second level occurs the execution of the simulation (the Model Swarm). Probes specified in

the Observer Swarm can gather events occurring in the Model Swarm, but the notion of behavioural event is not formally represented in the platform; the developer must implement this kind of observation in order to track such events. MadKit provides two ways for observing behavioural events: (i) the hook mechanisms; and (ii) agentified services managed, for instance, by the `OrganisationTrace` and `MessageTrace` agents. Hook mechanisms are generic subscribe-and-publish schemes that keep the subscriber agent informed about the invocation of kernel operations. The role of `MessageTrace` is to intercept calls of `sendMessage()` at the micro-kernel. The agent `OrganizationalTrace` traces all method invocations related to organisational abstractions (e.g. `jointGroup()`, `requestRole()`, see [16]).

B. Observe Cognitive Events: All platforms provide services to observe internal events. However, such services do not explicitly comply with models of cognitive reflectivity. In CORMAS the designer needs to implement methods in `SmallTalk` in order to observe the entities' internal attributes. In Swarm, this requirement is implemented through the interface `probe` and the `Observer` agents. This mechanism gives access to the agents' internal methods and variables at run time. The MadKit simulation engine, called synchronous engine, observes cognitive events through the `Watcher` agents and the `Probe` class. The `Watcher` agents are in charge of managing a list of `Probes`.

C. Provide Data Analysis: In CORMAS and Swarm there is a reasonable number of tools that fulfil this requirement. Meanwhile, in the current version of MadKit this requirement is not fulfilled. CORMAS allows the design of graphical charts with time-steps as the x-values and values returned by methods as the y-values. Additionally, two levels of data-recording are available. The first level tracks information from individual agents and the second from the agency. Swarm offers the `simtools` library to aid the process of data analysis, such as batch swarms that store information in files, generation of graphics and histograms. It also provides a very complete library to manage random number generators.

4 Conclusions

The requirement analysis that we have presented in this paper is a useful reference to define development principles for agent-based simulation platforms. The comparative analysis of CORMAS, MadKit and Swarm validated the requirements specification, which allowed us to characterize both the general structure and global behaviour of MABS platforms. Additionally, it was able to show that there is a reasonable consensus in regard to *technological*, *domain*, *development* and *analysis* facilities. Of course, some requirements are more developed in a platform than in others, but in general all platforms satisfied most requirements.

It is important to point out that in CORMAS and Swarm the distinction between behavioural and cognitive events is not completely and formally structured. In Mad-

Kit the observation of behavioural events is explicit, but the observation of cognitive events is not structured around models of cognitive reflectivity.

In our opinion, these limitations reflect a tendency in MABS to: (i) delegate in the designer the activities associated with structured data gathering; (ii) overlook the importance of integrating both behavioural and cognitive information to describe emergent phenomena. Indeed, none of these platforms fulfilled all requirements associated with exploration facilities (at least in a formal way), which reinforces our observations. Exploration facilities concern the active exploration of different results and qualitative concepts, which is the effective *raison d'être* of agent-based simulation. Such requirements are useful sources to indicate general principles that should be provided in the next generation of multiagent-based simulation platforms:

- Availability of observation and intervention mechanisms, for both behavioural and cognitive events, according to generic models of agent architectures and cognitive reflectivity; flexible management of observation and intervention windows, with respect to both individual and aggregate events;
- Availability of organisational models that should be able to manage topologies of multiple societies in a dynamic way. These models should be able to provide: (i) agentification of observation and intervention activities; (ii) agent primitives to launch different models of societies; (iii) dynamic creation of topologies of opaque and non-opaque observation spaces, which can be autonomously created by observer agents. This means that the topology of multiple societies in the simulation world can assume an emerging autonomous character from the human designer, as well as its different emergent points for opaque observation of social spaces;
- Availability of different ontologies to assist knowledge sharing, reuse, interoperability, simulation modelling and establishment of ontological commitments between simulation components.

References

1. Anderson J. and Evans M. (1995). A Generic Simulation System for Intelligent Agent Designs. *Applied Artificial Intelligence*, v.9, n.5, pp. 527-562.
2. Booch G, Rumbaugh J., Jacobson I. (1999). *The Unified Modelling Language User Guide*. Addison-Wesley.
3. Bousquet F., Bakam I., Proton H. and Le Page C. (1998). Cormas: Common-Pool Resources and Multi-Agent Systems. *Lecture Notes in Artificial Intelligence*, v. 1416, pp. 826-838.
4. Castelfranchi C. (1998). Simulating with Cognitive Agents: The Importance of Cognitive Emergence. In Proc. First. International Workshop on Multi-Agent Based Simulation (MABS'98), *Lecture Notes in Artificial Intelligence*, v.1534, Berlin: Springer-Verlag, pp. 26-41.
5. Ciancarini P. and Wooldridge M. (eds), Agent-Oriented Software Engineering, *Lecture Notes in Artificial Intelligence*, v. 1957, Berlin: Springer-Verlag, 2001.

6. CORMAS: Common-Pool Resources and Multi-Agent Systems - User Guide, <http://cormas.cirad.fr>, 2001.
7. David N., Sichman J.S. and Coelho H. (2002). Multiple Society Organisations and Social Opacity: When Agents Play the Role of Observers. In 16th Brazilian Symposium on Artificial Intelligence (SBIA'02), *Lectures Notes in Artificial Intelligence*, v.2507, Berlin: Springer-Verlag, pp.63-73.
8. David N., Sichman J.S. and Coelho H. (2002). Towards an Emergence-Driven Software Process for Agent-Based Simulation. In this Volume.
9. Decker K. (1996). Distributed Artificial Intelligence Testbeds. In O'Hare and Jennings, editors, *Foundations of Distributed Artificial Intelligence*, New York: John Wiley & Sons, pp. 119-138.
10. Ferber J. and Gutknecht O. (1998). A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems, In *Third International Conference on Multi-Agent Systems (ICMAS'98)*, IEEE Computer Society, pp. 128-135.
11. Gasser L. (2000). MAS Infrastructure Definitions, Needs, Prospects. In Infrastructure for Agents, MAS and Scalable MAS, *Lecture Notes in Artificial Intelligence*, v.1887, Berlin: Springer-Verlag, 1-11.
12. Gasser L. and Kakugawa K. (2002). MACE3J: Fast Flexible Distributed Simulation of Large, Large-Grain Multi-Agent System, In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002)*.
13. Gruber T. (1993). Toward Principles for the Design of Ontologies used for Knowledge Sharing. In Nicola Guarino and Roberto Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers.
14. Johnson P. and Lancaster A., *Swarm User Guide*, <http://www.santafe.edu/projects/swarm/swarmdocs/userbook/userbook.html>, 1996.
15. Lemaitre C. and Excelente B. (1998). Multiagent Organization Approach. In *Proceedings of II Iberoamerican Workshop on DAI and MAS*, Toledo, Spain.
16. MadKit, *Multi-Agent Development Kit*, <http://www.madkit.org>, 2002.
17. Marietto M.B., David N.C., Sichman J.S. and Coelho H. (2002). Requirements Analysis of Multi-Agent-Based Simulation Platforms. *Technical Report*, University of São Paulo.
18. Michael F., Gutknecht O. and Ferber J. (2001). Generic Simulation Tools Based on MAS Organization, In *Proceedings of Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'01)*.
19. Minar N., Murkhart R., Langton C. and Askenazi M., *The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations*, <http://www.santafe.edu/projects/swarm/overview/overview.html>, 1996.
20. Moss S., Gaylard H, Wallis S., and Edmonds B. (1998). SDML: A Multi-Agent Language for Organizational Modelling. *Computational and Mathematical Organization Theory*, v. 4, pp. 43-69.
21. Pfleeger S.L. (1991). Software Engineering. *The Production of Quality*, New York: Macmillan Publishing, 2nd edition.
22. Sichman J.S., Conte R. and Gilbert N. (1998). Multi-Agent Systems and Agent-Based Simulation, *Lecture Notes in Artificial Intelligence*, v.1534, Berlin: Springer-Verlag.
23. SimCog. *Simulation of Cognitive Agents*, <http://www.lti.pcs.usp.br/SimCog>.
24. Uschold M. and Jasper R. (1999). A Framework for Understanding and Classifying Ontology Applications, In *Proceedings of the IJCAI99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, Stockholm, Sweden.
25. Vincent R., Horling B. and Lesser V. (2000). An Agent Infrastructure to Build and Evaluate MAS: The Java Agent Framework and Multi-Agent System Simulator. In *Infrastruc-*

ture for Agents, MAS and Scalable MAS, Lecture Notes in Artificial Intelligence, v. 1887, Berlin: Springer-Verlag, pp. 102-127.

26. Zambonelli F. Jennings N. and Wooldridge M. (2001). Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. In *Int. J. of Software Engineering and Knowledge Engineering*, v. 11, n.3, pp. 303-328.