In: Proceedings of Multi-Agent Based Simulation Workshop, Bologna, Italy, 15-19 july, 2002.

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

Maria Bruno Marietto[1,*], Nuno David[2,†], Jaime Simão Sichman[1,‡] and Helder Coelho[3]

[1] Intelligent Techniques Laboratory, University of São Paulo, Brazil
Graca.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 work we propose a 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 Multi-Agent-Based Simulation (MABS), computational platforms are important methodological tools that aid the researcher in the 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 artificial intelligence, social psychology, social biology, economics and sociology. The interdisciplinary character of MABS is an important challenge faced by all researchers, 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, development and scientific research objectives are in increasing demand.

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,

* On leave from Dom Bosco Catholic University. Supported by FAPESP, Brazil, grant number 00/14689-0.
† Partially supported by FCT/PRAXIS XXI, Portugal, grant number BD/21595/99.
‡ Partially supported by CNPq, Brazil, grant number 301041/95-4.

verification and validation of models. If such topics and solutions are not well defined this 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 these solutions with respect to the 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 (e.g. [3,7,9,14,15,18]). 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 hard 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 [17] aims at three 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 to make a *comparative analysis* between different platforms that are presently available to the research community. The third goal is the specification, design and implementation of a platform complying with a subset of these requirements, with special focus on the simulation of cognitive agents. Regarding the first goal, a set of requirements guiding the specification of MABS platforms may be found in [12]. In this work we report our results with respect to the second goal, the comparative analysis. This paper is structured as follows. In section 2 we will present a set of requirements that characterizes both the general structure and global behaviour of MABS platforms. In section 3 we will present three well-known platforms and evaluate the adequacy of our reference model with a comparative analysis between those platforms. 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 point out 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 [6]. We identify four facilities that can be found in these computational systems: *technological*, *domain*, *development* and *analysis*. In this work, computational systems that present in 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 [12] we present a requirements specification for MABS platforms. As stated by [16], 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 [12] 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 nearly fifty (50) requirements, distributed between the five considered facilities. The requirements were classified in two categories, typical in requirements engineering: functional and non-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, without detailing the functionality of requirements.

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) 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:

- 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;
- Each functional requirement has a name. In order to connect requirements with UML use cases, each name is an active phrase, for instance “Launch Agents”.

In the rest of this section we will describe requirements clustered around the foresaid 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. It should provide libraries with the most usual scheduling techniques, like discrete-time simulation and event-based simulation.

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 considering how the absence of requirements may difficult, 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 can difficult the modelling and simulation of distributed 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 [8]). The first class, called *operational failures*, works with disturbances on 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 a simulation environment must be totally controlled, any 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 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, some of the agents' events should be notified to the simulator, which should update its local view (see [18]).

2.3 *Development* Facilities

Development facilities include mechanisms and tools to construct MAS within an agent-centered approach or organisation-centered approach.

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

B. Manage Communication Methods: 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. Manage 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 functions, activities or responsibilities. It should also be possible to define their interaction protocols. 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. Manage Multiple 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 through specific and pre-defined social events. Although the term society is an influential organisational metaphor in MAS, its concept is rarely specified as an explicit structural and relational entity [5]. 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 opaque 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 multiple societies. Questions of social opacity [5] between different societies must then be considered (see §2.5C).

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 deductions associated with a specific observer. This kind of question is found in [2], where the author points out the need to construct structured simulations in order to explain the emergence and "imergence" phenomena. According to [2], 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 [4] we show that this problem arises in MABS because the simulated *computational model* may not be complete, or even specified correctly, in relation to the intended *conceptual model* specification. Additionally, the *conceptual model* specification 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 results in the simulation that could in theory affect the intended conceptual model, the underlying feedback loop may have not been specified in the computational model, bringing the observed result to the realm of epiphenomenalism.

Nevertheless, it is possible to specify requirements that can help the researcher to observe objective simulation results. In [4] we view agent-based models as a set of *Agentified Entities* (AE) and the relations among them. Agentified Entities may be (i) atomic entities (e.g. an agent, an organisation); or (ii) aggregates specified with observation mechanics and interaction algorithms that manipulate atomic entities or other aggregates. The former are called *micro-AEs* and the latter are called *macro-AEs*. Macro AEs may differ in order of level, associated with different levels of aggregation. The AEs interact with each other through primitives that define the set of *behavioural* events specified in the model. 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 of behavioural events issued by micro-AEs on macro-AEs and vice-versa.

Figure 1 illustrates lines of information flow caused by events which can be observed

at different levels of abstraction. 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 invocations of behavioural events, where each AE acts as an information transformation function. Black points denote observation of behavioural events and internal states of AEs. Dotted lines denote control activity exercised by the simulator.

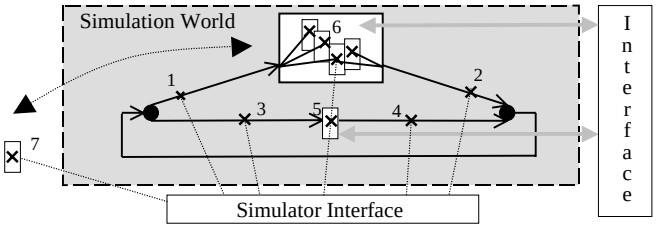

Figure 1 – Points of observation and intervention on behavioural and cognitive events.

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 points to observe behavioural events. Observation windows are given in the figure by points (1), (2), (3) and (4). Points (1) and (2) observe whole sets of behavioural events issued by AEs associated with a same macro-AE. Points (3) and (4) observe individual ones.

B. Observe Cognitive Events: Cognitive events are related to the agents' internal architectures. The observation of these 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 behavioural events on the recipient agent's mental states; or if the effect perceived by the agent who issues behavioural events 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 the figure, point (6) denotes the observation of whole sets of AEs associated with a same macro-AE. Point (5) and (7) denotes the observation of individual AEs. In order to provide structured means of observation, the platform should comply with another requirement, called cognitive reflectivity (see §2.5D).

C. Manage 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

These facilities emphasise the exploration character of simulations, in regard to exploration of different results and emerging qualitative concepts. This is important because most classic simulation programs and contemporaneous models of agent-oriented software engineering are not very concerned with the exploration of emergent, non-anticipated, structures. While most classic software processes are more concerned with *analysis* and *exploration of requirements*, the MABS software process is also concerned with *exploration of results* (see [4]). The exploration of different initial conditions, sequences of method invocation, mental states or assignment of variables is thus a crucial issue. These experiments can be facilitated if such conditions are allowed to change during the simulation, in-between simulation steps.

A. Intervene on Behavioural Events: The platform should offer mechanisms to select

specific points to intervene on behavioural events. The intervention should permit the *suppression*, *modification* or *creation* of behavioural events. For instance, one may want to modify the content or the intended recipient agent of behavioural events, or even to inhibit its arrival to the intended recipients. These experiments give the means to analyse functional effects in the simulation, independing from the agents' internal representations that originate other or those same events. In figure 2, points (1) and (2) intervene on whole sets of behavioural events issued by AEs associated with a same macro-AE. Points (3) and (4) refer to individual ones.

B. **Intervene on Cognitive Events:** The platform should offer mechanisms to intervene on the agents' internal mechanisms, for instance on the agents' beliefs or order of method invocation. This may alter the order of invocation and nature of behavioural events. Point (6) denotes the intervention on whole sets of AEs and point (5) on individual AEs. The platform should comply with the requirement §2.5D, cognitive reflectivity.

C. Manage Social Opacity: The problematic of social opacity analyses the organisational conditions under which the control of cognitive information transfer between agents in different societies (see §2.3D) is possible [5]. 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 simulation of agents that reason autonomously about the heterogeneity of different models of societies at various levels of observation. However, 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 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. Different models of cognitive reflectivity should be provided together with generic agent architectures, associated with requirement §2.3A. If the simulator infrastructure is based on MAS organisations, one can use opaque social spaces and a set of *monitor agents* that use cognitive reflectivity to observe or intervene on the simulated agents' cognitive events.

3 Comparative Analysis of MABS Platforms

We are now in conditions to present the platforms that will be subject to our 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, but 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 Dévelopment (CIRAD), France. The platform supports interactions between individuals and groups sharing renewable natural resources (see [3,11]). 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. Social entities are agents.

The Multi-Agent Development Kit (MadKit) platform is being developed by the Laboratoire d'Informatique, Robotique et Micro-Electronique of Montpellier, 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 [9]).

In the Santa Fe Institute, Chris Langton initiated the Swarm project in 1994, U.S.A. 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 [10,14]).

In Table 3.1 we present a comparative overview of CORMAS, MadKit and Swarm. The *exploration* facilities are 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.

Table 3.1: Comparison Between MABS Platforms.

	CORMAS	***MadKit***	***Swarm***
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 and applets	Agents can be launched as objects and 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 Communication Methods	Synchronous and asynchronous modes	Synchronous and asynchronous modes	Synchronous mode
Manage 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
Manage Multiple Societies	Not available	Groups do not quite resemble the concept of societies, but can be made alike	Multiple societies are swarms, allocated in hierarchical levels
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
Data Analysis	Rich set of tools	Not available	Libraries like simtools and random

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 [11] 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 [10,14].

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 [13], the specialization of the class `Activator` permits the customisation of different scheduling techniques for different sets of agents. The class `Scheduler` defines the agents 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, 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 hinders 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/C. Intentional Failures / Controlled and Non-Controlled Environments: N/A.

3.3 *Development* Facilities

Requirements related to development facilities are well structured in MadKit, since the platform is based on the Aalaadim organisational model [9]. 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 organisation 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 that help on 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.

B. Manage Communication Methods: CORMAS works with message passing and the users can define their own communication language. The class `Msg` defines a basic syntax with the attributes: `sender`, `receiver` and `symbol` [3]. 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 a `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 users can define their own communication language.

C. 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, in order to simulate interactions between different scales and organisational levels. The concept of group is implemented through the class `Group` and its subclasses (see [3]). In MadKit groups are atomic sets of agent aggregation. Each group has its own inner characteristics, like communication language, ontology, coordination techniques [9]. 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 `swarm` in different hierarchical levels are opaque to each other.

D. Manage 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 highly tied and created with specific inner organisational characteristics, like the roles an agent is allowed to play or the agents in the group with which he can communicate. 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 such societies (see §2.5.C).

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 expressible in the environment and what is subjectively represented in the agents' mind. In MadKit this separation is not entirely structured, but it is more flexible than in CORMAS and Swarm, partly due to it organisation-centered approach.

A. Observe Behavioural Events: In CORMAS the designer needs to implement methods in SmallTalk to operate the observation of behavioural events, providing sophisticated tools to observe those 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()`, see [9]).

B. Observe Cognitive Events: All platforms provide services to observe internal events. However, none provides this in a structured way, according to 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 operationalized with 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. Manage 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 the x-values as time-steps and the y-values with values returned by methods written by the developer. Additionally, two levels of data-recording are available. The fist level tracks information from individual agents and the second from the agency. Swarm offers the `simtools` library with classes that 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 presented in this work 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 and was able to characterize both the general structure and global behaviour of those 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 of them satisfied most of the requirements. It is important to point out that in CORMAS and Swarm the distinction between behavioural and cognitive events is not formally structured. In MadKit the observation of behavioural events is explicit but the observation of cognitive events is not structured around models of cognitive reflectivity. These limitations reflect a tendency in MABS to: (i) delegate to the designer the activities associated with structured data gathering; (ii) overlook the importance of integrating behavioural and cognitive information when explaining emergent phenomena. These limitations are reinforced when we observe that no platform fulfilled requirements related to exploration facilities. Such requirements concern the exploration

of different results and qualitative concepts, which is the effective *raison d'être* of agent-based simulation. Exploration requirements are useful sources to indicate general principles that should be provided in the next generation of multi-agent-based simulation platforms: (1) 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, in regard to both individual and aggregate events; (2) availability of organisational models that can manage multiple societies in a dynamic way. Such organisational 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 platform can assume an emerging autonomous character from the human designer with respect to the multiple society topology, as well as to its different points for opaque observation of social spaces and simulated agents.

References

1. Anderson J. and Evans M. A Generic Simulation System for Intelligent Agent Designs. *Applied Artificial Intelligent*, v. 9, n. 5, pp. 527-562, 1995.
2. Castelfranchi C., Simulating with Cognitive Agents: The Importance of Cognitive Emergence. In Proceedings of MABS, Springer Verlag, LNAI1534, pp. 26-41, 1998.
3. CIRAD. *Cormas: Commom-Pool Resources and Muli-Agent Systems - User Guide*, http://corma.cirad.fr, 2001.
4. David N., Sichman J.S and Coelho H., Towards an Emergence-Driven Software Process for Agent-Based Simulation. In this Volume.
5. David N., Sichman J.S. and Coelho H., Multi-Society Organisations and Social Opacity: When Agents Play the Role of Observers, To appear, see http://www.lti.pcs.usp.br/SimCog.
6. Decker K., Distributed Artificial Intelligence Testbeds. In O'Hare and Jennings, editors, *Foundations of Distributed Artificial Intelligence*, John Willey & Sons, pp. 119-138, 1996.
7. Gasser L. and Kakugawa K., MACE3J: Fast Flexible Distributed Simulation of Large, Large-Grain Multi-Agent System. In Proceedings of AAMAS 2002.
8. Gasser L., MAS Infrastructure Definitions, Needs, Prospects. In *Infrastructure for Agents, MAS and Scalable MAS*, LNAI1887, Springer-Verlag, pp. 1-11, 2000.
9. Gutknecht O., *MadKit Development Guide*, http://www.madkit.org, 2000.
10. Johnson P. and Lancaster A., *Swarm User Guide*, http://www.swarm.org.
11. Le Page C., Bousquet F., Innocent B., Alassane B. and Baron C., CORMAS: A Multiagent Simulation Toolkit to Model Natural and Social Dynamics at Multiple Scales. *Workshop "The Ecology of Scales"*, Wageningen (Pays-Bas), June, 2000.
12. Marietto M.B., David N., Sichman J.S., Coelho H., Requirements Analysis of Agent-Based Simulation Platforms. *Technical Report*, 2002, see http://www.lti.pcs.usp.br/SimCog.
13. Michael F., Gutknecht O. and Ferber J., Generic Simulation Tools Based on MAS Organization, Proceedings of *MAAMAW*, 2001.
14. Minar N., Murkhart R., Langton C. and Askenazi M., *The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations*, http://www.swarm.org.
15. Moss S., Gaylard H, Wallis S., Edmonds B., SDML: A Multi-Agent Language for Organizational Modelling. *Computational and Mathematical Organization Theory*, v.4, 1998.
16. Pfleeger S.L., *Software Engineering. The Production of Quality*, New York: Macmillan Publishing, 1991.
17. SimCog. *Simulation of Cognitive Agents*, http://www.lti.pcs.usp.br/SimCog.
18. Vincent R., Horling B. and Lesser V., An Agent Infrastructure to Build and Evaluate MAS: The Java Agent Framework and Multi-Agent System Simulator. In *Infrastructure for Agents, MAS and Scalable MAS*, LNAI1887, Springer-Verlag, pp. 102-127, 2000.