MIC*: A Deployment Environment for Autonomous Agents

Abdelkader Gouaïch, Fabien Michel, and Yves Guiraud

LIRMM, CNRS,
161 rue Ada,
34392 Montpellier Cedex 5, France
{gouaich, fmichel, yguiraud}@lirmm.fr

Abstract. This paper presents the MIC* model of autonomous agents deployment environment. A practical social software engineering framework based on AGR is also presented to show how MIC* is used to develop MAS applications.

1 Introduction

Multi-agent systems (MASs) are composed by autonomous agents (AAs) that evolve and interact in order to achieve their goals. What is implicit in this definition of MASs is where these AAs live. This containing place of AAs is identified by the generic term of environment. As Odell and colleagues have pointed out in [1], the environment defines the properties of the world in which an agent can and does function.

However, there are different concerns for the environment regarding the level of abstraction at which the attention is focused [2].

At the conceptual level, the environment defines the model of the AAs' world and the practical means by which they perceive and act on it to achieve their goals. At the implementation level, agents are necessarily embedded in a software system that offers them some computing facilities.

As Zambonelli and Parunak have noticed in [3], traditional software engineering approaches usually do not consider the environment at the implementation level as a primary abstraction. In the scope of this paper, the software system containing the AAs and defining their interactions is identified as the *deployment environment* (DE).

This paper relies on the idea that understanding and explicitly representing the DE is a crucial issue for MAS engineering. Moreover, this paper argues that the DE plays a fundamental role in order to guarantee the autonomy property. In fact, we will see that the *internal integrity* of AAs is an objective criterion that guarantees the autonomy at the implementation level.

As an example of DE, this paper presents MIC* (Movement, Interaction, Computation). MIC* is an algebraic model that is independent from both the conceptual and implementation models of the AAs. Hence, AAs are considered as black-boxes that sense and act through the DE by sending and receiving *interaction objects* (IOs). The interaction between the AAs is defined contextually in *interaction spaces* (ISs). The whole dynamics of the DE is seen as the composition of three kinds of functions: the movement, the interaction and the computation. MIC* is a DE which is defined at the implementation level. Consequently, MIC* has not to be confused with the modeling of the AAs' application-dependent world. For instance, agent-based simulations which have been developed using MIC* consider the situated environment as a particular active entity operating on the DE [4]. For instance, this entity is in charge of calculating the evaporating/aggregating/diffusing of pheromone, giving an operational semantics of simultaneous actions or environmental variables such as temperature and so on.

Finally, as a case study, this paper describes the implementation of a social framework which relies on traditional organizational concepts inspired by the AGR model [5].

The rest of the paper is organized as follows: Sect. 2 presents the backgrounds of the work; Sect. 3 discusses the autonomy property and induces the requirements that make necessary the explicit representation of the DE within MASs at the implementation level; Sect. 4 presents the MIC* DE; Sect. 5 shows how a social framework is built upon MIC*; Sect. 6 presents an application developed using this social framework; and finally Sect. 7 concludes and gives some perspectives.

2 Backgrounds

2.1 Multi-agent Systems

Few works tackle the general study of DEs. In this perspective, a lot of agent platforms have been developed and are available for the implementation of MASs [2]. However, these DEs are passive within the MAS and are often considered as basic middlewares used to *(i)* access computer resources and *(ii)* delivering messages to agents on the basis of predefined and fixed routing mechanisms. MIC* does not settle the interaction and routing mechanisms, it gives simply general requirements that should be instantiated for each particular MAS.

2.2 Coordination Media

The coordination medium can be considered as a persistent place where the interaction between the coordinating entities takes place. Linda [6] is an example of such a coordination medium where the entities coordinate their activities by writing and retrieving *tuples*. A tuple is a set of typed fields and values. Linda has inspired many other tuple-based coordination media such as Lime [7], Tuscon [8] and MARS [9]. The interesting feature of coordination media is the property of *generative interaction* [10]. This means that the interaction between entities is uncoupled in space and time. MIC* tries to offer the same property for the interaction among AAs. However, unlike Linda-like approaches, MIC* gives an explicit structure of the medium and defines its dynamics according to the MAS paradigm.

3 Why an Explicit Model of Deployment Environment Is Needed?

3.1 Implementing Autonomous Agents

This section uses Wooldridge and Jennings definition of an agent [11]:

> "*Perhaps the most general way in which the term agent is used is to denote a hardware or (more usually) software-based computer system that enjoys the following properties:*
> – *autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state [12];*
> – *social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language [13];*
> – *reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;*
> – *pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.*"

This definition specifies some features that a physical or software entity must fulfill to be considered as an AA. Still, this definition does not specify how to implement AAs. Consequently, developers may have their own interpretation of the presented features. In [14], Gouaïch identifies two main interpretations of autonomy in the MAS literature: autonomy as self-governance and autonomy as independence.

3.2 Autonomy as Self-governance

This interpretation is related to the definition proposed by Steels in [15]. Steels considers the autonomy feature from a biological point of view:

> "*It starts from the idea that agents are self sustaining systems which perform a function for others and thus get the resources to maintain themselves. But because they have to worry about their own survival they need to be autonomous, both in the sense of self-governing and of having their own motivations.*"

The concept of autonomy is thus regarded as a consequence of a survival instinct. For a software agent, it is a question of achieving its own goals while ensuring its functional requirements. Notably, an agent must be able to adapt itself with respect to a modification of the external environment. Castelfranchi [12] shares also this vision and defines an AA as a pro-active entity which has the ability to produce its own laws and to follow them.

3.3 Autonomy as Independence

This interpretation relates the autonomy feature to the social context of an AA. The *Social Dependence Network* (SDN) has been introduced by Sichman and colleagues in [16] to allow AAs to reason about their artificial society. Within SDN, the autonomy concept is used to evaluate the level of the social dependence. Three forms of autonomy are distinguished. An agent is *a-autonomous* for a given goal according to a set of plans, if there is a plan in this set that achieves the goal and every action in each plan belongs to its capabilities. An agent is considered as *r-autonomous* for a given goal according to a set of plans, if there is a plan in this set that achieves the goal, and every resource in each plan belongs to its resources. Finally, an agent is *s-autonomous* when it is both *a-autonomous* and *r-autonomous*. According to this definition, an agent is autonomous for a particular goal if it does not depend for resources or actions on another agent.

3.4 Internal Integrity: An Objective Criterion for Autonomy

Sichman and colleagues define autonomy as being independent on actions and resources from other agents. On the other hand, [12, 15, 17] define agent's autonomy as a behavioral characteristic. From a software engineering perspective, the latter interpretation is more useful and generic since it does not imply to study the MAS social structure. The autonomy is only related to individual characteristics. However, it still remains a subjective point of view because it relies on how the behavior of an agent is evaluated. So, as Weiss and colleagues have pointed out in [18], objective implementation criteria are necessary to define the autonomy of a software agent. We propose the *internal integrity* as an objective criterion to implement AAs [14].

The internal integrity is a programming constraint that considers an AA as a bounded system which internal dynamics and structure are neither controllable nor observable directly by an external entity. In fact, if the AA's software structure is accessed or modified by another entity, the decisional process and behaviors may be altered. Since the decisional process of an AA has to be entirely determined only by its own perception and behaviors, the internal integrity becomes a sine qua none condition to implement AAs.

3.5 Agent Deployment Environment: Ensuring Internal Integrity

The internal integrity criterion also raises some issues with respect to the implementation of MASs: on one hand, the internal integrity has to be taken into account to guarantee the autonomy; on the other hand, the AAs are interacting entities that need to act and modify the perceptions of other agents. Since these perceptions are included within the boundaries of the AAs, this contradicts the internal integrity statement. In other words, the problem is to enable the interaction between AAs which boundaries do not intersect. To avoid this paradox, the DE needs to be a non-agent entity that manages and carries out the interactions.

The next section presents MIC* as an example of a DE that guarantees the internal integrity of the AAs while enabling their interactions.

4 MIC* Algebraic Model

4.1 Introduction to the MIC* Model

In order to fulfill the presented requirements on autonomy, the AAs have to be considered as bounded black-boxes. Thus, no assumption is made on their internal structure. Consequently, the DE only considers the observable processes such as the interaction. The interaction process is independent from the AAs conceptual and implementation models. In fact, heterogeneous AAs are able to interact at least if they agree on a common interaction language or ontology. Within MASs, the word 'interaction' is misused and often refers to a communication process. Communication is defined as exchanging information between several locations; while interaction goes further and assumes that the exchanged information modifies the state of the communicating entities. To be exchanged, information is usually encoded using explicit carriers. Within MIC* these carriers are reified as *interaction objects* (IOs).

For instance, a researcher's ideas can be encoded as words and sentences in an explicit scientific paper which represents the explicit information carrier. Other human agents are able to read this IO and, depending on their competences, to decode the contained information.

Once the paper has been written and published, the emitting agent does not have control on the ongoing communication processes that occur.

For instance, Socrates is still in a communication process with other human agents centuries after his death. Having this intuition about IOs, it would be interesting to look further in their structure. The first abstraction is to define an empty IO that carries no information. For instance, an empty paper is an IO that does not carry any information but just meta-information: it is a paper and it is empty.

The IOs can also be aggregated. For instance, the proceedings of the conference is an IO represented as an aggregation of more elementary IOs. Consequently, IOs naturally have a monoid structure $(\mathcal{O}, +)$ with the composition law $+$ and identity element 0. In an aggregation, we do not want to consider the order as an additional information. So, no matter the order of the IOs in an aggregation, one has to be able to interpret them similarly. This makes the composition operator $+$ commutative.

Now let us consider a situation where a poor-quality paper is rejected by the program committee of a conference and accepted by a national workshop program committee. This IO never reached the perceptions of other agents in the first case and interacts with them in the second case. So, the interaction process are contextually defined. This introduces the concept of *interaction spaces* (ISs). Hence, ISs define a local context for interactions among IOs. Notice that the interaction processes within MIC* only involve IOs and is completely independent from the AAs.

The AAs have coordinates, in terms of IOs, in all ISs. When an agent is not 'present' in a certain IS, its representation is equal to the empty IO 0; when an agent is present in a certain IS its representation differs from 0.

The (logical) mobility of an AA is defined as the movement of its IOs among ISs. In order to easily define this notion of mobility, we introduce negative IOs. Hence, an AA moves outside an IS when its representation is reduced to 0. This can be expressed as $x + (-x) = 0$. So, negative IOs are defined as being IOs that reduce other IOs under the composition law $+$. So, the IOs structure is no more a commutative monoid but a commutative group $(\mathcal{O}, +)$. The group structure is also used in order to define the composition of several MIC* DEs. Thanks to this on-the-fly composition property, MASs for open and ubiquitous contexts are easily modeled and implemented [19].

The MIC* structure $\mathcal{T} = \mathcal{O}^{(\mathcal{A}\times\mathcal{S})} \times \mathcal{O}^{(\mathcal{A}\times\mathcal{S})}$ is composed by two matrices that are described as follows:

1. The outbox matrix: the rows of this matrix represent agents $i \in \mathcal{A}$ and the columns represent the ISs $j \in \mathcal{S}$. Each element of the matrix $o_{(i,j)} \in \mathcal{O}$ is a representation of the agent i in the IS j. This is the only way for an agent to exist and operate in the MAS. So, the elements of this matrix model the means that enable an agent to perceive and influence the universe in a particular IS. Notice that the means used to perceive the universe are distinguished from the result of the perception. The perception results are placed in the inbox matrix. When $o_{(i,j)} = 0$, the agent i neither influences nor perceives the universe in the IS j: agent i does not exist in IS j.
2. The inbox matrix: the rows of this matrix represent agents $i \in \mathcal{A}$ and the columns represent the ISs $j \in \mathcal{S}$. Each element of the matrix $o_{(i,j)} \in \mathcal{O}$ represents the result of the perceptions of the agent i in the IS j.

Each element, or term, T of $\mathcal{T}$ is represented as:

$$T = \underbrace{\begin{pmatrix} \underbrace{\begin{bmatrix} \underbrace{[o_1]_a}_{\textbf{(C)}} \\ \vdots \end{bmatrix}_s}_{\textbf{(B)}} & \cdots \end{pmatrix}}_{\textbf{(A)}} \underbrace{\begin{pmatrix} \underbrace{\begin{bmatrix} \underbrace{[i_1]_a}_{\textbf{(G)}} \\ \vdots \end{bmatrix}_s}_{\textbf{(F)}} & \cdots \end{pmatrix}}_{\textbf{(E)}}$$

(A) : the outbox matrix ; (B) : the IS 's' ; (C) : the outbox of agent 'a' ; (E) : the inbox matrix ; (F) : IS 's' ; (G) : the inbox of agent 'a'.

4.2 *MIC** Dynamics

An element $T \in \mathcal{T}$ is an instantaneous snapshot of the DE state. Within all potentially functions defined from $\mathcal{T}$ to $\mathcal{T}$, MIC* considers three classes which have special semantics for MASs:

Interaction (ϕ): From an external point of view, two AAs are considered as interacting when the perceptions of an agent are influenced by the emissions of another. Consequently, interaction functions modifies the perception results of an agent (defined in the inbox) according to its perception means and others influences (both defined in the outbox) within a defined IS. The set of all interaction functions is represented as ϕ.

Movement (μ): The mobility of an agent is defined as the mobility of its IOs among different ISs. During a movement no IO is created nor lost. In fact, this is an interesting feature to prevent incoherent duplications by guaranteeing that an AA actually disappears from its original IS and appears in its destination IS. The set of all movement evolutions is represented by μ.

Computation (γ): The computation is an internal process of AAs. The only way to observe that an AA has conducted a computation is when it changes autonomously its outboxes within ISs. To avoid confusion between computation and movement, after a computation, AAs conserve their presence. In other words, an agent is not allowed to appear (respectively to disappear) suddenly in an IS when it was not present (respectively present) before the computation. Besides, agents are rational entities that change their emissions according to their perceptions. So, an agent consumes its perceptions in order to make a computation. This is expressed in MIC* by resetting the inbox of the computing agent to 0. The set of all computation evolutions is represented by γ.

The core idea is that *(i)* the DE dynamics is discrete and *(ii)* any state of the DE is reached from the initial state by a sequence of functions that may be of three classes: (M)ovement, (I)nteraction, and (C)omputation (MIC*).

4.3 Building MAS Deployment Environment with MIC*:

The formal concepts presented above has been implemented as software structures offering a development library for the designers of DEs. To complete the design and implementation of a DE, the designer has to provide the followings:

– IOs type description: the IOs have been used: *(i)* to encode and carry information, *(ii)* to define the perceptions of AAs in ISs, *(iii)* to define the influences of AAs in ISs, and finally *(iv)* to define the movement of AAs among ISs. A typing system of IOs has been introduced to describe the fields contained in an IO and to provide a semantics. The types of IOs have also been used to improve the performances of MIC*, especially when using the dynamics operators. Hence, the operators, which are typed functions, are executed only when the matching IOs are present within the IS. The DE designer has to provide the description of the different IO types used within the MAS and their possible hierarchical relationships. The introduced type system supports multiple inheritance.
– Interaction spaces: the MIC* library offers a default IS where AAs are initially located. The designer has to define its own application specific ISs.

Dynamics Operators. The dynamics of MIC* is realized by the following operators that are defined for ISs:

- Interaction operators: a couple of IOs is passed to the interaction operator, namely the *sensor* and *effector*. The interaction operator returns the *interaction result*. An IS may contain zero or more interaction operators. Consequently, the interaction is defined between the IOs and not between the AAs. The AAs have to set their outboxes to the correct sensors in order to perceive the effectors of other AAs according to the defined interaction operators. The type information is used to match interaction operators with corresponding IOs present in the IS.
- Movement operators: the movement among the ISs is decomposed in two operators which are combined. In fact, each IS defines a set of *movement out* operators that specify which IOs are allowed to get outside the IS; and a set of *movement in* operators that defines which IOs are allowed to enter the IS. A path is created between two ISs when the types of their corresponding movement out and in operators match. If a path is found among two ISs, the IOs may move from the source to the destination IS.

5 Building a Social Framework Upon MIC*

MIC* only offers a generic and low level abstraction of a DE. To build real world applications, one has to provide a higher level engineering framework. This section presents a social framework. The idea is that MAS designers only deal with social concepts which are automatically translated to MIC* concepts.

5.1 Presentation of the Social Framework

The presented social framework is deeply inspired by the AGR model [5]. The MadKit [20] platform already implements the AGR model; here we explore another implementation using only MIC* primitives.

The social abstractions presented by AGR are briefly described as follows:

- (A)gent: an agent may play one or several roles and may be member of one or several groups;
- (G)roup: a group is a collection of roles and consequently a collection of agents that play these roles. The interaction among the agents can occur only when they are located within the same group;
- (R)ole: a role is an abstraction that represents a function or a service within the society; agents playing the role fulfill the desired service.

At this stage, let us sketch a preliminary mapping between AGR and MIC*. As shown in Fig. 1, the group concept may be modeled as an IS: the IS concept may be seen as a logical location where a collection of agents interact. Besides, the agents may move across groups; this is similar to moving across ISs. The agent concept of AGR naturally corresponds to MIC* AAs. Still, there is not a one to one mapping between these concepts (see Sect. 5.2). The role concept is considered as an IO within MIC*. In fact, when an agent plays a certain role,

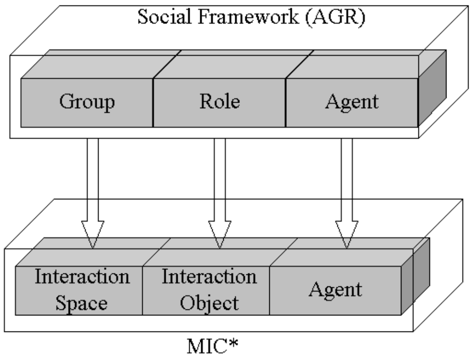

Fig. 1. Mapping between AGR concepts and MIC* concepts

it publishes an IO that describes itself as playing this role. Consequently, other agents can identify its social function and interact with it. The implementation of the AGR model using MIC* is explained in more detail in Table 1.

Two interaction schemes are considered for the social framework:

1. The role-level interaction schema: messages are delivered to agents only by knowing their roles. This mechanism allows implementing one-to-many communications and the discovery of agents' identities by knowing only their roles.
2. The agent-level interaction schema: messages are delivered to agents by knowing their exact identity. This mechanism implements one-to-one communications.

5.2 Implementation of the Social Framework

Interaction Objects. Fig. 2 presents the types of IOs used in the social framework:

– `Message`: the `Message` type represents IOs used to exchange information encoded as a content. This is the base-type of all other interaction related types; it contains a single field, `content`, that represents the exchanged information.

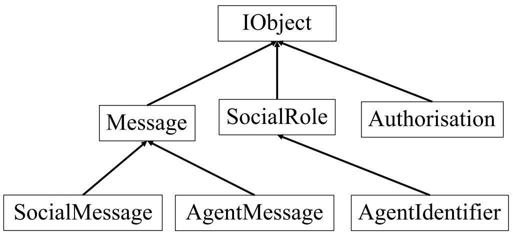

Fig. 2. Type hierarchy of IOs used in the social framework

– `SocialMessage`: the `SocialMessage` type represents exchanged messages for the role-level interaction schema. The fields of this type are: `sender-role` that represents the role of the sender and `receiver-role` that represents the role of the receiver. This type inherits the `content` field from the `Message` type.
– `AgentMessage`: the `AgentMessage` type represents exchanged messages at the agent-level interaction schema. This type fields are: `sender-agent-id` that represents the identity of the sender; `receiver-agent-id` that represents the identity of the receiver; `sender-role` that represents the role of the sender; `receiver-role` that represents the role of the receiver. This type inherits the `content` field from the `Message` type.
– `SocialRole`: the `SocialRole` type represents roles which are played by the AAs. This type defines only a single field `role-id` that represents the unique identifier of the role.
– `AgentIdentifier`: the `AgentIdentifier` type represents the identity of an agent. In fact, since AAs do not have access to the structure of others, they have to explicitly publish their identity. This type defines only a single field `agent-id` that represents the unique identifier of the agent; it also inherits the `role-id` field from the `SocialRole` type.
– `Authorisation`: the `Authorisation` type is used to control group access using movement operators. An agent is allowed to enter an IS by presenting the correct `Authorisation` instance. The `Authorisation` contains the name of the played role, namely the `played-role` field; and the `certificate` field that represents a signature confirming that the agent is allowed to play this role.

Interaction Operators. Two interaction operators are defined in order to model the interaction schemes:

– Role-level interaction operator: this operator is defined among `SocialRole` and `SocialMessage`. A `SocialRole` interacts with a `SocialMessage` only and only if the receiver role of the `SocialMessage` is the same as the role-id field of the `SocialRole`. This is expressed algorithmically as:

```
1: function RoleLevelIOP::interaction(sensor,effector)
   Require:   sensor is instance of the SocialRole type
   Require:   effector is instance of the SocialMessage type
2:     if sensor['role-id'] == effector['receiver-role'] then
3:         return effector
4:     else
5:         return 0                                    ▷ No interaction.
6:     end if
7: end function
```

– Agent-level Interaction Operator: the agent-level interaction is defined between `AgentIdentifier` and `AgentMessage`. An `AgentIdentifier` interacts with an `AgentMessage` only and only if the id of the receiver is the same as the id of the agent. This is expressed algorithmically as follows:

```
1: function AgentLevelIOP::interaction(sensor,effector)
   Require:   sensor is instance of the AgentIdentifier type
   Require:   effector is instance of the AgentMessage type
2:     if sensor['agent-id'] == effector['receiver-agent-id'] then
3:         return effector
4:     else
5:         return 0                                          ▷ No interaction.
6:     end if
7: end function
```

Movement Operators. The groups are modeled as ISs. Consequently, each IS is associated with a set of roles. To enter the IS, an agent has to play a role that belongs to this set. To realize these movements, the group-entrance operator allows agents to enter inside an IS. The agents have to present an `Authorisation` that describes the played role. On the other hand, the group-leaving operator allows agents to leave the IS.

Interaction Spaces. Besides the default IS defined by MIC*, each group is represented by an extension of MIC* IS, namely the *social interaction space*. Each social IS is defined with a set of authorized roles; the role-level and agent-level interaction operators; and the group-entrance and group-leaving operators.

The Autonomous Agents. Within MIC*, the AAs may have simultaneous activities. For instance, a single AA can sense its surrounding environment and affect it simultaneously. To realize this simultaneity, several MIC* agent entries are used. For instance, Fig. 3 shows this schema where two MIC* agent entries are associated to a single AA: the *sensor* entry and *effector* entry.

From the AA perspective, the sensor entry is dedicated for sensing the universe. Consequently, IOs that perceive the universe are placed in the outbox

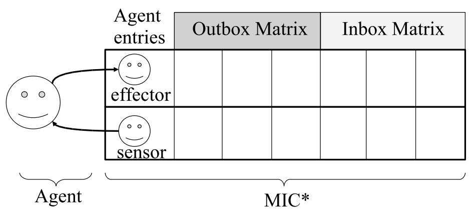

Fig. 3. A single AA have several entries within the MIC* deployment environment: an entry dedicated to sense the universe, i.e. the sensor entry; and an entry dedicated to affect the universe, i.e. the effector entry

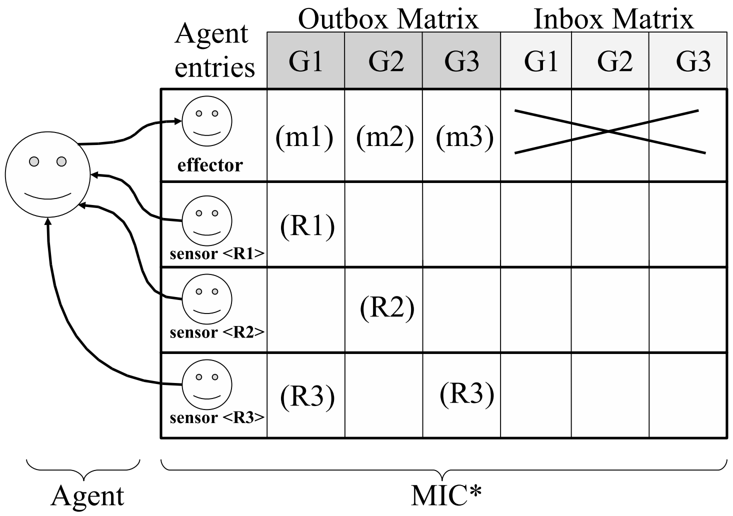

Fig. 4. A social autonomous agent owns an effector entry to affect the universe, and several sensor entries representing its roles

matrix, and the result of their interaction is placed in the inbox matrix. On the other hand, the effector entry is dedicated to affect the universe; consequently, IOs to be perceived by other agents are placed in the outbox matrix and, in this case, the inbox matrix is not used (marked with X, see e.g. Fig. 4). For the MIC* environment, the agent's sensor and effector entries are considered as independent agents; the AA is responsible for making this couple of agents behaving as a single entity. This seems similar to the Holonic approach that considers a set of agents as an single agent [21]; still, here we argue that a set of agents that have been conceived to behave as a single entity can build a global agent.

To represent the fact that an AA plays several roles in groups, the mechanism presented above is extended such that each AA is associated to an effector entry and zero or more sensor entries. Each sensor entry represents a played role. Figure 4 gives an example of an AA that plays three roles R_1, R_2 and R_3. This AA has a single effector to send messages, for instance this agent is sending three messages simultaneously m_1, m_2 and m_3 in three groups G_1, G_2 and G_3. This agent plays simultaneously several roles within the same group: R_1 and R_3 in the group G_1; and plays the same role in several groups: R_3 in G_1 and G_3.

5.3 Mapping Table Between AGR and MIC*:

Finally, by considering the presented concepts, the complete mapping among the AGR concepts and MIC* is described by Table 1.

Table 1. Mapping between AGR concepts and commands to MIC*

AGR	**MIC***
Creation of a group named *x*	This is performed by creating an IS identified as *x*
Deletion of the group *x*	This is performed by deleting the IS identified as *x*
The agent *a* joins the group *x*	The agent *a*'s entries (effector and sensors) enter the IS *x* using the *Authorisation* IO. The involved movement operator is the group-entrance operator.
The agent *a* leaves the group *x*	The agent *a*'s entries (effector and sensors) leave the IS *x* using the group-leaving movement operator.
The agent *a* acquires the role *r*	The agent *a* acquires the *Authorisation* IO with a valid certificate.
The agent *a* plays the role *r* within the group *g*	The agent *a* moves inside the IS *g* using the *Authorisation* IO; when this agent is inside the IS, he changes its outbox to the *SocialRole* IO; now this agent is perceived by the others as playing the role *r*.
The agent *a* stops from playing the role *r* within the group *g* for a short period	The agent *a* has only to change how the others perceive him; so it changes its *SocialRole* IO to another one; when the agent wants to resume playing the role *r*, he puts back its *SocialRole* IO.
The agent *a* drops the role *r*	The agent *a* leaves all ISs where he was perceived as playing the role *r*.
The agent a_1 playing the role r_1 sends a message to the agent a_2 playing the role r_2 within the group *g*	The message to be sent is an instance of *AgentMessage* type. The agent a_1 puts this message as a computation of its effector in the *g* IS; then the interaction operator performs the actual interaction among the agent a_1 effector outbox and the agent a_2 sensor corresponding to the role r_2. The result of the interaction is found in the inbox of the agent a_2. In this case, the agent-level interaction operator is used.
The agent a_1 playing the role r_1 sends a message to any agent playing the role r_2 inside the group *g*	The message to be sent is an instance of *SocialMessage* type. The agent a_1 puts this message as a computation of its effector in the *g* IS; then the interaction operator performs the actual interaction among the agent a_1 effector outbox and all sensors corresponding to the role r_2. In this case, the role-level interaction operator is used.

6 Example of Application: Ubiquitous Web

The *Ubiquitous Web* is an application that emulates the use of the web in a mobile and ubiquitous environment. The purpose of this section is to present how such an application has been modeled and built using the social framework and to demonstrate its deployment on a simulated 3D virtual world.

6.1 Organizational Modeling:

Systemic Functions. The goal of the application is to emulate the navigation and access of html-based services for ubiquitous environments. There are two systemic functions:

1. Web navigation and access of services: the software system is divided in two main parts, namely the server and client modules. The server module is responsible for delivering web-pages that describe the offered services and forms. The client is responsible for translating the user's commands into requests to the server and displaying the html pages.
2. Discovery of services: the server module of the system is also responsible for delivering a human readable description of the offered services. On the other hand, the client is responsible for discovering all accessible services and for retrieving their description.

Organisational Structure

Roles

1. WebServerRole: this role responds to the requests of agents playing WebClientRole. A WebClientRole may request a html presentation; or request a service. When requesting a service, the WebClientRole agent can deliver the parameters which have been set by the user.
2. WebClientRole: this role represents the intermediary function between the final user and the WebServerRole. The functions of this role are:
   (a) request a particular html presentation from the WebServerRole;
   (b) correctly layout the html presentation to the user;
   (c) request services from the WebServerRole by sending the parameters of the service as imported by the user.
3. ServiceDiscoveryServerRole: the main function of this role is to deliver a human readable description of the offered service; and to deliver an access point where to contact the actual service provider.
4. ServiceDiscoveryClientRole: the main function of this role is to check the presence of ServiceDiscoveryServerRole agents and to retrieve their description and the service's access point.

Groups

1. WebGroup: this group holds agents that play WebClientRole and WebServerRole roles.
2. ServiceDiscoveryGroup: this group holds agents that play ServiceDiscoveryServerRole and ServiceDiscoveryClientRole roles.

6.2 Simulation of Ubiquitous Environments

In order to experiment with the application, a simulator has been developed for ubiquitous environments using computer games technologies. The goal of this

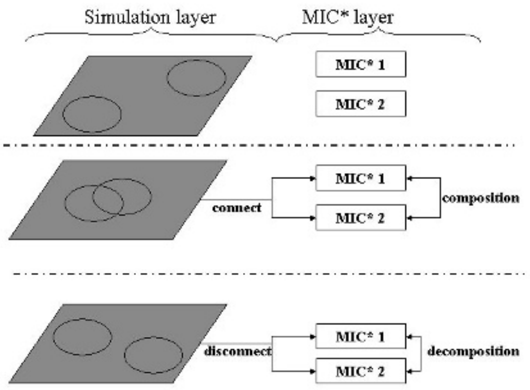

Fig. 5. MIC* DEs are composed and decomposed according to the avatars' communication areas

simulation is to emulate a physical world where the user, represented by an *avatar*, can move and interact with the deployed services which are also represented as avatars. Behind each avatar an entire MAS is running: this includes the AAs and the corresponding MIC* DE. Each avatar has a communication area; when the avatars' communication areas overlap, their corresponding MIC* DE are on-the-fly composed. Similarly, when the avatars' communication areas do not intersect, their corresponding MIC* DE are disconnected. This process is shown graphically by Fig. 5. The user has a 'First Person Shooter' (FPS) perspective and can move around in the virtual world. Figure 6 presents the main views:

– Situation A: there is no service in the immediate surroundings of the user.
– Situation B: the user perceives a service, but the service is too far away to establish a composition of the DEs.
– Situation C: since a communication link can be established, the MIC* DEs are composed. Consequently, AAs located in both deployment environments can interact.

When the user leaves the building of the service (after situation C), the MIC* DEs are immediately decomposed. Consequently, the AAs cannot interact. These are the realistic properties of the ubiquitous environment and the applications have to handle them. Thanks to the on-the-fly composition property of MIC* DEs, the constraints on communication links do not disturb drastically the software systems. In fact, these constraints are handled explicitly in the developed models. For instance, a disconnection is not very different from a silence of an AA that has decided to not reply to external stimuli.

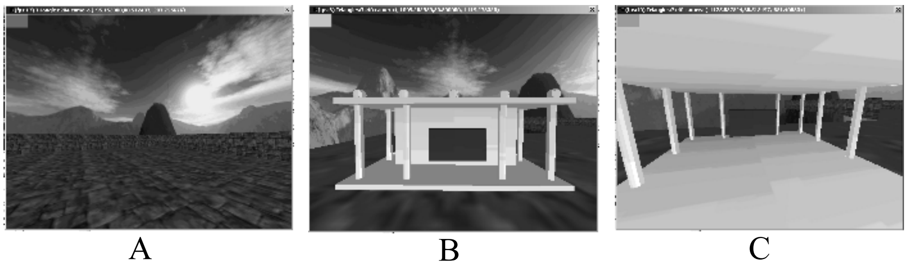

Fig. 6. First-Person-Shooter (FPS) perspectives in the simulator

6.3 End User Graphical Interface

The user interacts with the client module through the web browser presented by Fig. 7:

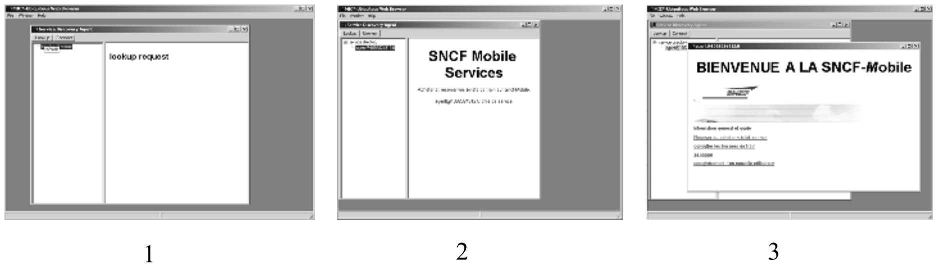

Fig. 7. Ubiquitous web browser GUI

- Situation 1: the agent that plays the role of ServiceDiscoveryClientRole has not discovered any service yet. An empty list is presented to the user. This corresponds to situations A and B of Fig. 6.
- Situation 2: the agent that plays the role of ServiceDiscoveryClientRole has discovered some services by interacting with the agent that plays the role of ServiceDiscoveryServerRole in the ServiceDiscoveryGroup group. The list of the available services is presented to the user. This corresponds to situation C of Fig. 6.
- Situation 3: the user has now a web-like interaction with the service. The involved agents are those playing the WebClientRole and WebServerRole roles. This also corresponds to situation C of Fig. 6.

7 Conclusion

This paper has argued that the DE is a key concept for agent-oriented engineering, since it guarantees the autonomy of the agents while it defines their interactions. As an example of such DE, MIC* has been presented.

The notion of DE separates the concerns of MAS engineering. In fact, the engineering of agents is completely separated from the engineering of DEs. The DE is the common structure offered to different developers to deploy AAs and make a global system which functions emerge from the interactions of the individuals.

MIC* offers some interesting features such as the implementation of the internal integrity for AAs; the generative interaction and the on-the-fly composition. These features have provided the basis for engineering open software systems in complex and unpredictable environments such as ubiquitous environments.

However, MIC* has to provide more elaborated control and trust functions. Currently, we are exploring the control of coordination and interaction protocols by MIC*. Hence, MIC* monitors the agents' conversations with regards to interaction and coordination protocols. Any AA that challenges these protocols is identified by the DE and other AAs are prevented from its influences. Consequently, the AAs are offered a normed DE where they can collaborate with autonomous partners.

References

1. Odell, J., Parunak, H.V.D., Fleischer, M., Breuckner, S.: Modeling agents and their environment. In Giunchiglia, F., Odell, J., Weiss, G., eds.: Agent-Oriented Software Engineering (AOSE) III. Volume 2585 of Lecture Notes on Computer Science., Springer, Berlin (2002) 16–31
2. Weyns, D., Parunak, H.V.D., Michel, F., Holvoet, T., Ferber, J.: Environments for multiagent systems: State-of-the-art and research challenges. In Weyns, D., Parunak, H.V.D., Michel, F., eds.: Environments for Mutiagent Systems. Volume 3477 of Lecture Note in Artificial Intelligence LNAI., Springer (to appear, 2005)
3. Zambonelli, F., Parunak, H.V.D.: From design to intention: signs of a revolution. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM Press (2002) 455–456
4. Michel, F.: Formalisme, méthodologie et outils pour la modélisation et la simulation de systèmes multi-agents. PhD thesis, Université Montpellier II (2004)
5. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In Paolo Giorgini, Jrg P. Mller, J.O., ed.: Agent-Oriented Software Engineering IV: 4th International Workshop, Aose 2003. Lecture notes in computer science LNCS, Springer Verlag (2003) 185–202
6. Gelernter, D., Carriero, N., Chandran, S., Chang, S.: Parallel programming in linda. In: Proceedings of the International Conference on Parallel Programming. (1985) 255–263
7. Picco, G.P., Murphy, A.L., Roman, G.C.: Lime: Linda meets mobility. In: International Conference on Software Engineering. (1999) 368–377
8. Omicini, A., Zambonelli, F.: The tucson coordination model for mobile information agents. 1st Workshop on Innovative Internet Information Systems (1998)
9. Cabri, G., Leonardi, L., Zambonelli, F.: Reactive tuple spaces for mobile agent coordination. Lecture Notes in Computer Science **1477** (1998) 237–247
10. Gelernter, D.: Generative communication in linda. ACM Transaction od Programming Languages and Systems **7** (1985) 80–112
11. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review **10** (1995) 115–152

12. Castelfranchi, C.: Guarantees for autonomy in cognitive agent architecture. In: Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents, Springer-Verlag New York, Inc. (1995) 56–70
13. Genesereth, Ketchpel: Software agents. Communications of the ACM **37** (1994) 48–53
14. Gouaïch, A.: Requirements for achieving software agents autonomy and defining their responsibility. In: The First International Workshop on Computational autonomy - Potential, Risks, Solutions (autonomy 2003). (2003)
15. Steels, L.: The biology and technology of intelligent autonomous agents. Robotics and Autonomous Systems **15** (1995)
16. Sichman, J.S., Conte, R., Castelfranchi, C., Demazeau, Y.: A social reasoning mechanism based on dependence networks. In Cohn, A.G., ed.: Proceedings of the Eleventh European Conference on Artificial Intelligence, Chichester, John Wiley & Sons (1994) 188–192
17. Luck, M., d'Inverno, M.: A formal framework for agency and autonomy. In Lesser, V., Gasser, L., eds.: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, CA, USA, AAAI Press (1995) 254–260
18. Weiss, G., Rovatsos, M., Nickles, M.: Capturing agent autonomy in roles and xml. In: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, ACM Press (2003) 105–112
19. Gouaïch, A., Guiraud, Y., Michel, F.: Mic*: An agent formal environment. In: the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), session on Agent Based Computing ABC'03. (2003)
20. Gutknecht, O., Ferber, J., Michel, F.: Integrating tools and infrastructures for generic multi-agent systems. In: Proceedings of the fifth international conference on Autonomous agents, AA 2001, ACM Press (2001) 441–448
21. Parunak, H.V.D., Odell, J.: Representing social structures in uml. In: Agent-Oriented Software Engineering II. Volume 2222 of Lecture notes in computer science LNCS., Berlin, Springer (2002) 1–16