Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2015, Article ID 731031, 11 pages
http://dx.doi.org/10.1155/2015/731031

*Research Article*

# Organizational Gameplay: The Player as Designer of Character Organizations

**Luciano R. Coutinho, Victor M. Galvão, Antônio de Abreu Batista Jr., Bruno Roberto S. Moraes, and Márcio Regis M. Fraga**

*Departamento de Informática (DEINF), Universidade Federal do Maranhão (UFMA), Avenida dos Portugueses 1966, Bacanga, 65080-805 São Luís, MA, Brazil*

Correspondence should be addressed to Luciano R. Coutinho; luciano.rc@ufma.br

Received 6 February 2015; Accepted 26 April 2015

Academic Editor: Ali Arya



Looking at the ways in which players interact with computer games (the *gameplays*), we perceive predominance of character-centered and/or microcontrolled modes of interaction. Despite being well established, these gameplays tend to structure the games in terms of challenges to be fulfilled on an individual basis, or by thinking collectively but having to microcontrol several characters at the same time. From this observation, the paper presents a complementary gameplay in which the player is urged to face collective challenges by designing *character organizations*. The basic idea is to make the player structure and control group of characters by defining *organizational specifications* (i.e., definitions of roles, collective strategies, and social norms). During the game, commanded by the player, artificial agents are then instantiated to play the roles and to follow the strategies and norms as defined in the organizational specification. To turn the idea into practice, the paper proposes an abstract architecture comprising three components or layers. This architecture is materialized in a proof of concept prototype that combines the Minecraft game server, JADE agent platform, and MOISE+ organizational model. Variations and possibilities are discussed and the proposal is compared to related work in the literature.

## 1. Introduction

Much of the fun involved in modern computer games is due to their extraordinary capacity of simulating aspects of our reality and/or imagination. In the diverse genres, such as *first person shooters* games (e.g., Halo and Call of Duty), *third person action/adventure* games (e.g., God of War and Tomb Raider), *platformers* (e.g., Super Mario Bros. and Sonic), *two-player fighting/sports* games (e.g., Fight Night Round and FIFA Soccer), *racing* games (e.g., Midnight Club and Need for Speed), *real-time strategy* games (e.g., Age of Empires and Command & Conquer), *open world/sandbox* games (e.g., Minecraft and The Sims), and *massively multiplayer online game* (e.g., World of Warcraft and Star Wars Galaxies), we can cite games in which the player gets the illusion of being immersed in a detailed and verisimilar virtual world. Commonly, in these simulated worlds the player assumes the control of a character (his/hers *avatar*) and interacts with diverse *nonplayer characters* (NPCs) or other *player-controlled characters* (other players' *avatars* in multiplayer games). Then, he/she is confronted to a series of challenging situations comprising missions to fulfill, rules to follow, decisions to make, territories to explore or to conquer, and so forth. According to [1], the fun emerges from this when the player succeeds to comprehend and to master the general patterns of action and interaction underlying the challenges faced in the virtual world. In this sense, the more a game is capable of portraying aspects of the player's experience as gradually learnable patterns, the greater the chance of it pleasing its users.

Notwithstanding the achievements of the modern game technology, especially concerning 3D animated graphics and sound effects that have produced games featuring high-quality enjoyable virtual worlds, we observe that the aspects of control and interactivity between players' avatars and NPCs (the *gameplay*) occur mainly on an individual and

nonscriptable basis. By this we mean that, in general, the player actively controls one avatar and interacts with the NPCs issuing proper commands. For example, in first person shooter games, the player assumes the identity of a soldier. The mission can be to assault an enemy base leading a special troop. During the mission, the player does his best in controlling his avatar while the other soldiers of the troop, which are NPCs, follow the movements by performing preprogrammed attack routines. The player neither sees nor directly commands the attack strategies used by his fellows; he/she simply assumes that the routines codified on the NPCs by the game designer are effective in making the troop to accomplish the mission. Another example can be given in real-time strategy games. In this genre, the player also commands troops in combat against enemies. The difference is that now the player constantly has to issue commands to make the NPCs to behave in this or that way. The attack strategy is enacted by the player which has to have great dexterity and multitask abilities to microcontrol all of them at the same time.

What we miss in these and related examples from the diverse genres of games is a gameplay more focused on the collective rather than the individual character, more focused on explicitly stated rather than microcontrolled collective strategies, and more focused on player-defined or player-editable rather than preprogrammed nonaccessible collective strategies. By these we mean a mode of control and interaction in which the player can be aware of the collective and social aspects of the game by creating, seeing, or modifying the *role hierarchy* and *team behavior* of a collection of characters at run-time. Take, for instance, the troop command scenario. The game could feature one or more basic strategies composed by a given number of soldiers in the troop, each soldier in a position, performing some role. Our idea is that the player can modify this troop organization if desired. He could create new roles, designate a different number of soldiers to the roles, establish the group coordinate behavior for assaulting the enemy's base, and so forth. Having done this organizational setup the player can concentrate on performing the role of his choosing while indirectly controlling the others characters (NPCs) in his group. As another example consider an open world/sandbox game, a game in which the player is supposed to explore a territory, find food, gather raw materials, build houses, villages, and so forth, all of this in collaboration with other avatars and/or NPCs. At first, the players could try all alone to explore the world and do the needed tasks. As they meet one another, they discover common needs and try intuitive ways of doing joint activity. After a while of experimentation, the players pass to a phase in which they could try to structure the work by coming up with collective plans or organized ways of doing things. In the process, inevitably, they are led to reflect in challenges such as: How do we organize ourselves to do the job? What roles should be played? How do we divide labor? How do we coordinate? How many of us are necessary for a task?

We believe that offering a gameplay experience where the players are encouraged to think about and practice with these issues is a valuable effort. We note that some genres, mainly collective sports games (e.g., soccer and basket), naturally lead to a group structuring/planning mindset like the one we have described. Actually, some games in this genre feature controls for the user to play the coach of the team, being able to choose from a list or to define his own team organization. However, by the genre's own nature, the user is driven to well-known predefined team configurations, consecrated by the practice. Sports games aside, the major opportunity we see is to turn the gameplay into a more encompassing learning experience, something that can teach the player more than just individual skill patterns, something with the potential to offer other kinds of challenges to the players, challenges of coordination, cooperation, and organization.

Motivated by these observations, we have set a research project in which the final aim is to develop what we call an *organizational interface for computer games*, or *organizational gameplay* for short. Our scope is computers games in which the player gets immersed in a virtual world where the challenges involve the interaction with other characters to realize joint activities. The organizational gameplay we envision is intended to complement, not to replace, the existing gameplay. The idea is to provide to the player visual representations and editor facilities for *character organizations*. What we call "character organizations" comprises two complementary aspects: on the one hand, the specification of role hierarchies and joint activities plans for dividing and coordinating tasks among characters of a group, on the other, the instantiation of roles and plans at run-time for controlling NPCs on behalf of the player. With these we expect to offer to the players a gameplay experience similar to the ones previously described.

Our approach takes advantage of recent developments in *autonomous agents* and *multiagent systems* technology [2, 3]. We note that the notion of an agent as a software component exhibiting *autonomy*, *social ability*, *reactivity,* and *proactiveness* [4] is a suitable conceptualization for characters of computer games [5]. Accordingly, the whole virtual worlds rendered in computer games can be seen as multiagent systems (MAS). Concerning the organizational gameplay, a key feature of agent technology is the existence of modeling languages and run-time infrastructures for designing and implementing *organization centered* MAS (OC-MAS) [6–8]. OC-MAS is a class of MAS where the agents' autonomy is regulated/controlled by means of explicit organizational structures (e.g., role hierarchies, standard procedures, interactions protocols, norms, etc.) imposed to the agents at run-time to ensure coherent collective behavior towards common goals. Ultimately, the idea is to import and adapt some of the concepts, notations, and infrastructures from OC-MAS to bring about our view of character organizations.

In this paper, our aim is to present the core idea of the organizational interface for computer games together with the implementation of a proof of concept prototype. In Section 2, we first introduce the basic concepts and developments we are using from autonomous agents and MAS technology. Then, in Section 3, we describe our general proposal for the organizational gameplay, contrasting it with the existing gameplay. After, in Section 4, we present and discuss a first prototype implementation of the whole proposal which was done in the context of Minecraft, a popular open

world/sandbox multiplayer game. In Section 5, we compare our work with similar work found in the literature. In Section 6, we present our final considerations and future work.

## 2. Background

Autonomous agents and multiagent systems have emerged in the broad area of *Distributed Artificial Intelligence* [9]. In the last years, with the growing perception that agents and MAS "offer a new and often more appropriate route to the development of complex systems, especially in open and dynamic environments" [3, page 204], these concepts were incorporated by other research communities, notedly *Software Engineering* [10, 11].

*2.1. Autonomous Agents.* The general idea of *agent* is perceived in virtually all areas of human knowledge where the notions of *individual*, *environment,* and *community* do exist. From a Software Engineering viewpoint, agent can be defined as a computer program (or software component) that is design to perceive and act autonomously on a given environment. Here the distinguishing property is *autonomy*. Among the possible meanings [12], we consider autonomy as the ability of deciding by itself when and how to act. Thus, in principle, an autonomous agent should be capable of choosing by itself what to look at and what to do in its environment or of refusing to perform some action when asked, in order to accomplish some goals it was designed for.

Furthermore, if an agent exerts its autonomy balancing goal-directed and reactive behavior in an effective way it can be called an *intelligent agent* [4]. *Goal-directedness* or *proactiveness* occurs when the agent takes the initiative of perceiving, deciding, and acting to fulfill predefined goals. *Reactivity*, on the other hand, means deciding and acting solely in response to perceived events in the environment. This distinction has led to the development of two main architectural styles known as *deliberative agents* (e.g., belief-desire-intention architectures) and *reactive agents* (e.g., subsumption architecture). In practice, autonomous intelligent agents usually are designed and implemented by mixing elements of these two architectural styles.

*2.2. Multiagent Systems.* A fourth desirable characteristic is *social ability*. Agents with social ability can get together and perform tasks beyond the reach of any of them individually. A collection of socially able agents gives rise to a *multiagent system* (MAS).

As long as agents are autonomous and designed to pursue particular goals, social ability implies more than interaction by message exchange within the MAS. Agents with conflicting agendas are expected to compete. Agents with reconcilable agendas are supposed to negotiate and/or cooperate. In this way, a crucial point in the development of MAS is how to reconcile the autonomy of the agents with interactions that lead to some desired global behavior, or, more broadly, "how to obtain from local design and programming, and from local actions, interests, and views, some desirable and relatively predictable/stable emergent result" [13].

Given this issue, some researchers have proposed to divide the MAS approaches into two major classes: *agent centered MAS* (AC-MAS) and *organization centered MAS* (OC-MAS) [14]. In AC-MAS, as the name suggests, the development efforts are focused on the design of the agents (bottom-up development). The aim is to come up with an agent architecture that enables not only a right balance between proactiveness and reactivity, but also between local agent goals and global MAS purpose. This kind of approach was common in the early years of the agent and MAS technology and has its roots in attempts to mimic swarm behavior observed in species like insects, birds, and so forth. In the last decade, with the growing interest in the development of complex systems, especially in open and dynamical environments [3, 11], AC-MAS have been superseded by OC-MAS.

In OC-MAS, *human organizations* are used as a proper metaphor for engineering complex open MAS. Human organizations, whose typical examples are corporations, clubs, armed forces, and so forth, are *collectivities* pursuing *specific goals* and exhibiting *formalized social structures* [15, page 26]. Goals are specific to the extent that they are explicit and clearly defined. Social structures are formalized in such a way that patterns of structuring and behaving (such as *roles*, *role relations*, *procedures*, *protocols*, *norms,* etc.) are precisely specified regardless of personal traits and relations of any individual person part of the organization. Thus, by conceiving an MAS as an organization, or more generally, as a bigger system formed by several *agent organizations*, the basic idea is to promote "social order" in a top-down fashion. The idea is to have the agents' actions and interactions governed by formalized "social structures," defined and enforced by mechanisms outside the agents, in order to enable the MAS as a whole to fulfill its purpose.

In the context of complex open systems, AC-MAS are not a good approach because they rely solely on the agent being specially created to interact with the others in an orderly fashion. But this can not be assumed on an open system where the components can be heterogeneous and possibly created at different times by different stakeholders. On its turn, OC-MAS are a better fit in this case because the system is designed to have some components, apart from the agents, which instantiate the expected social order. At run-time any agent that enters the organization is supposed to abide by the formal rules, procedures, and structures underlying the agent organization. If not, they can be penalized or even expelled from the organization.

*2.3. Organizational Design.* The design and implementation of OC-MAS rely on two technological developments. The first one is modeling languages for specifying agent organizations. The other is feasible ways of translating the specifications into separate components for controlling and coordinating the agents at run-time.

Regarding modeling languages, there are various proposals in the literature [7, 14, 16, 17]. Commonly, they

provide the designer with some *modeling constructs* and *notation* for expressing patterns of structuring and behaving that characterize an agent organization. Four basic kinds of modeling constructs can be identified [18]. Firstly, there are concepts to represent *statical aspects* of the *internal structure* of the agent organization. In almost all proposals this is the primary modeling concern expressed in terms of *roles*, *relationships*, and *groups*. An agent, to be part of the organization, should play some available role (*position*) in a group. Roles define expectations with regard to the agent's abilities and responsibilities. Relationships (like *has-authority-over*, *can-communicate-with,* or *depends-on*) further refine general expectations and define new constraints the role-playing agents should observe. Groups are clusters of roles or of role-playing agents that serve to partition the organization into related areas.

Secondly, there are modeling constructs to specify the *functional behavior* of agent organizations. Generally, the functional behavior is expressed in terms of *goals*, *missions*, *tasks,* and *hierarchical plans*. Goals are declarative representations of the units of work comprising the purpose of the organization. In some proposals, the goals are partitioned into coherent subsets called missions; a posteriori, missions are assigned to roles. Tasks are operational descriptions of how to achieve goals. Tasks can be individual, when in the reach of one agent playing a role, or collective, when their performance needs the coordinated effort of several role-playing agents. Collective tasks are commonly scripted as collective procedures in the form of hierarchical plans, that is, task decompositions trees where the leaves represent individuals tasks, the internal nodes represent the composition of finer-grained tasks to realize coarser-grained tasks, and the root represents a global task for achieving some major goal of the organization.

A third basic category of modeling constructs is formed by concepts to depict the *structural behavior* of agent organizations. The structural behavior concerns an orthogonal view of the realization of tasks and goals. Rather than hierarchically decomposing tasks into subtasks, the idea is to define *interaction protocols* and *phase transition graphs* to specify how goals can be achieved over time. An interaction protocol is a prescription of sequences of message exchange among roles to perform some task or achieve some goal. State transition graphs are state machines that break whole functioning of the organization into major *phases* (or *scenes*) and valid transitions between phases triggered by events.

Lastly, there are modeling constructs to further restrict, regulate, and interrelate internal structure (roles), structural behavior (interactions), and functional behavior (procedures). Here, the basic notions are the concepts of *norms* and *sanctions*. Norms are the definition of *permissions*, *obligations*, *prohibitions*, and so forth. For instance, a norm statement could be as follows: an agent playing a given role is obliged to fulfill an specific mission; to fulfill the mission, the agent is allowed to use a given hierarchical plan when in a given phase but is prohibited to use it when in another phase. Sanctions are penalties imposed to the agents after the infringement of norms.

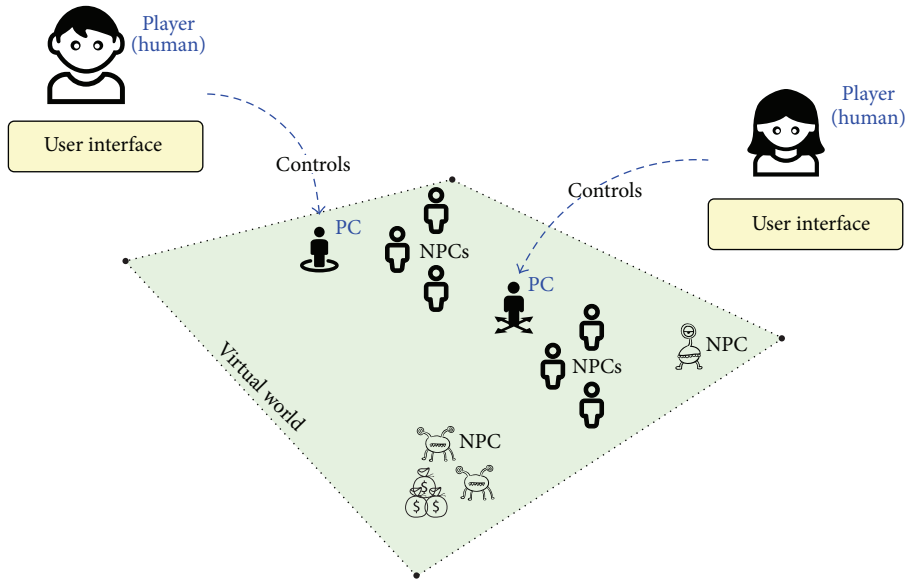
*2.4. Platforms and Infrastructures.* Regarding the implementation of agents and agent organizations, it can be done by means of *agent platforms* supplemented with *organizational infrastructures*.

Broadly, an agent platform is a computational environment offering several facilities for developing and running software agents. It can provide the developer with skeletons for programming reactive and deliberative agents, special programming languages for expressing the action-decision strategy (means-ends analysis and practical reasoning) of the agents, and services for dynamical identification, localization, and transport of messages within an MAS. Presently, many of the platforms follow some of the standards proposed by the *Foundation for Intelligent Physical Agents* (FIPA (http://www.fipa.org/)), an IEEE Computer Society organization that promotes interoperability standards for agent-based technology. As examples of agent platforms we cite JADE (*Java Agent DEvelopment Framework* [19]), Jason (*Java-based AgentSpeak interpreter used with SACI for multiagent distribution Over the Net* [20]), and Madkit (*MultiAgent Development KIT* [14]).

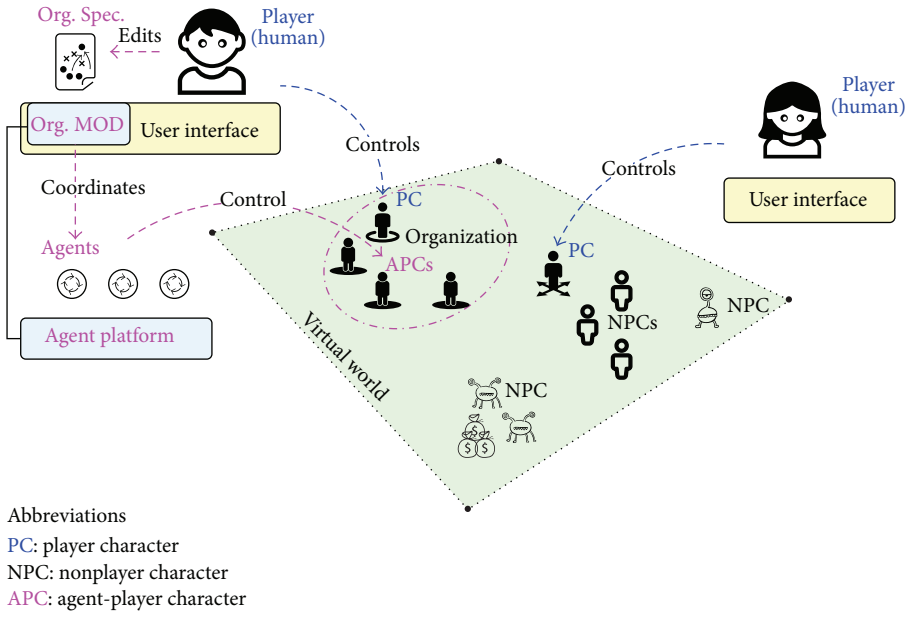
Primarily, agents platforms are supposed to provide tools for developing plain agent-based applications. In the case of OC-MAS, the practice has been to combine them with organizational infrastructures. An organizational infrastructure is a software framework and/or run-time facilities for producing fully functional agent organizations from organizational specifications. First the MAS designer specifies the desired organization by using an organizational modeling language supported by the infrastructure. Then he submits the specification to the infrastructure which generates/instantiates special components for coordinating and controlling the agents at run-time; the subsystem formed by these components can be called *organization run-time environment*. To become a member of the organization, the agents will have to ask the organization run-time environment for a role to play and commit themselves to the realization of the corresponding missions, rights, and duties. As example of organizational infrastructure we cite MOISE+ (*Model of Organization for multI-agent SystEms* [17]), which can be used in combination with Jason. Another example is Madkit, which is not only an agent platform but also an OC-MAS development kit based on the concepts of *Agents, Groups, and Roles* (AGR model [14]).

## 3. Organizational Gameplay

Autonomous agents and MAS are convenient notions for conceptualizing characters and virtual worlds of computer games. Player controlled characters are in essence avatars, agents whose autonomy and social ability are provided by human players. Nonplayer controlled characters (NPCs) are artificial agents; their autonomy and social ability are provided by the artificial intelligence subsystem of the game engines. In this sense, the virtual world of a computer game is a heterogeneous MAS. It comprises software agents (NPCs) and is open for human agents by means of avatars (Figure 1(a)).




(a) Traditional gameplay




(b) Proposed gameplay

Figure 1: Organizational interface for computer games.

In this section, we describe how to exploit this conception in the definition of an abstract architecture from which to develop the organizational gameplay discussed in Section 1.

*3.1. Traditional Gameplay.* Similarly to MAS, a major challenge in developing virtual worlds for computer games is how to design coherent joint activity. We refer to games where characters are supposed to form groups with common interests. A basic configuration is one avatar as leader and several NPCs as companions, as illustrated in Figure 1(a). Usually, the challenges the groups face are beyond the abilities of each character separately. Thus, to act as verisimilar groups, the characters should coordinate their joint efforts.

Traditionally, coordination is preprogrammed or microcontrolled. As pointed out in Section 1, either the NPCs follow routines defined by the designer, or the player continually sends them commands to act in this or that way. In both

cases, the player is not encouraged to think about the group as an explicitly organized collective. The player knows that the NPCs should help him to achieve the missions but is unable to act as a real team leader in the sense of devising an explicit collective plan to structure and coordinate the actions of the group. With this, the player misses the opportunity of focusing on the learning social, collective patterns, rather than just individual skills.

Here, we can make a parallel to AC-MAS and OC-MAS. Normally, the design and control in computer games occur in an agent centered basis. Avatars and NPCs, as agents, are programmed to locally react to events or conditions in the game in such a way that a globally coherent behavior does emerge. Mostly, the design and control are not organization centered. Avatars and NPCs are not designed to perceive, reason, and proactively act according to organization specifications outside the agents. As long as the computer game is supposed to be open only to avatars, not to external artificial agents, and the player is not asked to reason about and control the group of characters as explicitly organized collectivities, the AC-MAS design is a reasonable choice. Otherwise, the OC-MAS approach presents itself as a more appropriate alternative.

*3.2. Proposed Gameplay.* To complement the traditional gameplay, we argue for an organization centered mode of coordination and control. The basic idea is to stimulate the player to think about the game as a series of challenges that demand not only individual skills but also organized collective work. The player is then asked to assume the leadership by devising formal social structures (i.e., role structures, collective plans, etc.) for achieving definite objectives, what could be done from scratch or by modifying and combining preexistent social structures. The intent is to enable the coordination and control of NPCs by means of these player-defined social structures. Clearly, our proposal is in essence the very idea of OC-MAS design recasted as a gameplay experience.

In Figure 1(b) we illustrate the proposed gameplay. Compared to Figure 1(a), the first significant enhancement is the possibility of the player to edit *organizational specifications*. These are written in an special purpose visual language that comprises some of the modeling constructs used in the organizational design of MAS. A second major enhancement consists in turning the NPCs into *agent-player characters* (APCs). APCs are artificial agents instantiated by the players, as needed, to assume the control of characters on their behalf. APCs should coordinate their joint efforts according to the organization specification defined by the player. Together, avatars and APCs will form what we call *character organizations*, which replace the groups formed by avatars and NPCs in the traditional gameplay.

Note that, by means of avatars, the player still can participate in the action on an individual basis. And, at the same time, by means of character organizations, the player now has the chance of reasoning and acting collectively without the need of constant microcontrol. Lastly, also note that character organizations do not eliminate NPCs altogether. Some of the challenges to be faced by characters organizations could be to defeat enemies programmed as NPCs, or even to compete with other groups formed by avatars and NPCs.

*3.3. Main Building Blocks.* The scheme of Figure 1(b) also conveys the general architectural pattern we put forward for actually developing the proposed organization centered gameplay. It comprises three major modules or layers: a *front-end module* for specifying and managing character organizations, a *middle layer* in which agents and organizations are instantiated, and a *back-end game server* in which the virtual world and characters are simulated.

The front-end module is part of the graphical player interface. It is the module with which the player can view and edit the social structure of character organizations. As mentioned, this is supposed to occur by using a special visual language. Depending on the game and desired level of detail, the visual language can cover different ranges of modeling concerns. At minimum, it can offer graphical notations for representing role structures and hierarchical plans. With this, the player can define how character groups should be composed, in terms of roles to be played, and how the groups should proceed to face the challenges, in terms of collective plans involving the roles. A more comprehensive language could go beyond roles and plans and provide additional notations to express interaction protocols, transition graphs, norms, and sanctions, virtually covering all the modeling concerns discussed in Section 2.3. Obviously, the more rich the language is, the more difficult it becomes to the player to properly use it, and the more difficult it is to implement agents and agent organizations able to interpret it. Therefore, concerning the visual language used in the front-end, a balance among expressiveness, usability, and implementability should be struck. Moreover, to promote usability, we stress that the game designer ought to provide the player with default social structures. By seeing, copying, and modifying these predefined specifications, the player is guided and invited to build his/her own specifications of character organizations.

After specifying the social structure of a character organization, the player should command the instantiation of agents to play the defined roles. Agents, and the necessary organization run-time environment, form an intermediate layer between the player's front-end and the actual simulation of the virtual world and characters. Here we identify the main technical challenges. Firstly, a platform for the development of agent-based games will be needed. Some game engines already provide modules for agents; an alternative could be to adapt existing agent platforms to interact with the back-end game server. Secondly, the available agent module or platform should be augmented with an organizational infrastructure. As discussed in Section 2.4, the organizational infrastructure is the element responsible for interpreting organizational specifications and implementing the run-time environment for coordinating and controlling agent organizations. Lastly, appropriate agents should be developed; agents that can access the organization run-time environment, assume roles indicated by the player, and properly command characters on the virtual world simulation.

The back-end game server represents the components responsible for simulating the virtual world and characters of the game. According to the standard practice, these can be implemented by means of *game engines*. Game engines are high-level platforms for game development [21]. They consist of a collection of modules for easing the process of creating 2D or 3D scenarios, simulating physical interactions, animating characters, and even doting NPCs with some level of learning and intelligent behavior. For our purposes, we assume that some of the characters simulated on the game server are controlled by external commands. These characters are not autonomous agents per se but surrogates (avatars) for the game players. As long as we have stated that the human players will instantiate artificial agents outside the virtual world to control characters within the virtual world, we can see these agents also as game players. For this reason we have named their characters as agent-player characters (APCs).

## 4. Proof of Concept

Front-end, middle layer, and back-end can be instantiated in various ways. For game developers, an ideal scenario would be to have an integrated game engine with which one could develop not only the virtual world but also agents, agent organizations, and the organizational front-end. As a proof of concept, we have implemented a prototype by combining three separate pieces of software: Minecraft game server, JADE agent platform, and MOISE+ organizational model (Figure 2).

*4.1. Minecraft Game.* Minecraft (https://minecraft.net/) is an open world/sandbox game. It features a 3D cube-based virtual world divided into *biomes* such as deserts, jungles, and snowfields. Primarily, there is no specific goals for the players to accomplish, who are free to choose how to play. Players, controlling an avatar, may wander around the various terrains, break, and gather *blocks of materials* such as stones, ores, water, and wood. Using these material blocks, they can build diverse structures like houses, buildings, roads, villages, and so forth. During the game, players encounter different NPCs called *mobs*. Mobs include animals (cows, pigs, and chickens, for instance), villagers (person-like characters), and hostile creatures (such as spiders, skeletons, and zombies). The player may hunt the animals for food, negotiate materials and goods with villagers, and combat the creatures. In sum, there are three basic game modes: *survival*, *creative,* and *adventure*. In survival mode, the player must look for natural resources in the environment to stay alive. These include materials to craft tools, weapons and shelters for protecting against creatures, and food to satiate hunger. In creative mode, the players are provided with plenty of resources to build anything they want. They can fly freely around the virtual world, do not suffer damage, or feel hungry. The player is stimulated to pursue large building projects. In adventure mode, the players are supposed to explore custom worlds created by other players. The gameplay is similar to survival mode, with restrictions and adventures introduced by the players.

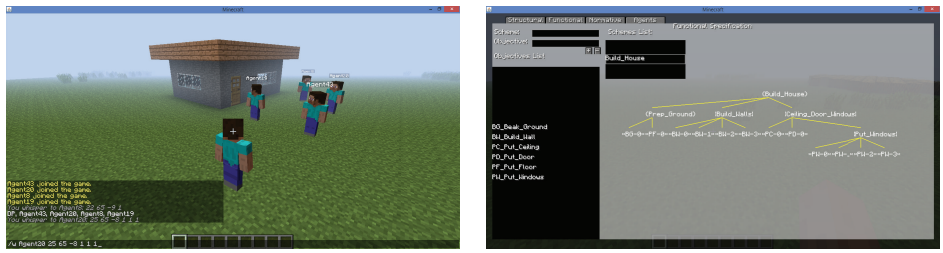
We have chosen Minecraft as the virtual world of our proof of concept prototype for three main reasons. First and foremost, it is an open world where the players are naturally encouraged to propose challenges and build things of their own. This is in accordance with our view that the gameplay experience should actively help the player to reason about useful behavioral patterns. Implementing the proposed organizational gameplay, we intend to expand this reasoning towards collective endeavours. Second, Minecraft has multiplayer support by means of game servers. Players can run their own servers on a local area network configuration or use remote servers. In this way, the players join a virtual world via a client interface, by which they issue commands to control their avatars. And this complies with our assumption that the some characters on the game server should be controllable by external commands, the hook for attaching agents as external players of characters.

Lastly, Minecraft allows the creation of game modifications (*mods*). These can range from simple additions of textures to large reformulations on the gameplay. By using this feature, we have implemented a basic organizational interface for Minecraft (Figure 2(b)). This interface enables the player to view and edit specification of character organizations following the MOISE+ organizational model. The interface also allows the player to instantiate agents, running on a container of the JADE platform, that are able to connect to a given Minecraft game server, join the virtual world of the game, and play specific roles of the specified character organization (Figure 2(a)).

*4.2. JADE Agents.* JADE (*Java Agent DEvelopment framework* [19]) is an open source agent platform that complies with FIPA specifications. It provides a run-time environment where agents can be executed, extensible class libraries for developing agents, and run-time tools for monitoring and managing the platform. The run-time environment is divided into containers. The containers may be instantiated in several hosts thus forming a distributed agent platform.

JADE was chosen for matters of convenience. It is based on Java, the same language in which Minecraft and MOISE+ are implemented, and offers a rich and mature environment for creating and running MAS applications. Using JADE, we have developed an agent architecture based on a particular perceive-decide-act life cycle. On the one hand, each agent must control an avatar on the Minecraft game server; on the other, it ought to consult the organization run-time environment to coordinate with the other agents. With this in mind, we structured the agent architecture around two types of behavior, one responsible for maintaining the connection to the game server; this behavior, called *w-behavior*, enables the agent to perceive, decide, and act on the virtual world. The other, called *o-behavior*, enables the agent to query and update the organization state. By alternating o-behaviors and w-behaviors, the agents become organization controlled artificial collaborators.

*4.3. MOISE+ Organizations.* O-behaviors take into account the primitives of an *organizational model*. By organizational



(a) Minecraft virtual world: human player and four JADE agents following MOISE+ specification

(b) MOISE+: organizational specification editor

Figure 2: Proof of concept prototype based on Minecraft, JADE, and MOISE+.

model we mean both the modeling language and the organizational infrastructure used to specify and operationalize agent/character organizations. As mentioned, we adopt MOISE+ in our proof of concept prototype.

In MOISE+ (*Model of Organization for multI-agent SystEms* [17]), organizational specifications are divided into three parts called *structural*, *functional,* and *deontic* specifications. The structural specification covers the statical aspects of the internal structure of agent organization in terms of *roles* and *groups* definitions, and *role-role*, *group-group,* and *role-group* relationships. The functional specification describes how to achieve global goals related to the main purpose of an organization. This description is done by means of *social schemes* and *missions*. In essence, a social scheme is a tree where the root is a *global goal*, the internal nodes represent *subgoals*, and the leaves stand for atomic goals that are supposed to be done by individual agents. A node together with its children means a *plan*. The plan consists in achieving the goal related to the parent node by combining the fulfillment of the goals related to the children. There are three operators for combining goals into plans: the *sequence* (i.e., fulfilling all the children goals, one after another, in a given order), the *choice* (i.e., fulfilling one of the children goals, chosen by the agents), and *parallel* (i.e., performing all the children goals at the same time). Missions are coherent sets of goals that can be assigned to the same agent by means of one of its roles. This assignment is done in the deontic specification. The deontic specification covers the normative aspects of the organizational design. It consists in the definition of *permissions* and *obligations* connecting roles to missions. In this way, an agent that joins a MOISE+ organization by assuming a given role must (or is permitted to) carry out all the goals forming the missions related to the role by means of an obligation (or permission) relationship. And the achievement of missions by role playing agents is to be synchronized by instantiating and following the plans embedded in a given social scheme.

In Figure 2(b), a fragment of a social scheme for building a house is shown. The global goal, or purpose of the organization, is then named `Build_House`. For fulfilling this goal, the major plan is to divide the work into three consecutive steps. First, do the groundwork (the subgoal `Prep_Ground`); second, build the walls (subgoal `Build_Walls`); and third, install the ceiling, door, and windows (subgoal `Ceiling_Door_Windows`). These subgoals are further divided into atomic goals: `Prep_Ground` is split in breaking the ground (`BG-0`) and putting the floor (`BG-0`), done in sequence; `Build_Walls` is broken down in building four walls (`BW-0, BW-1, BW-2, BW-3`), in parallel; and `Ceiling_Door_Windows` consists in building the ceiling (`PC-0`), putting a door (`PD-0`), and four windows (`PW-0, PW-1, PW-2, PW-3`), all done in parallel. (In the editor, a goal enclosed by {} or || represents the application of the sequence or parallel operator, resp.). These goals are gathered into missions, like `resource collection`, `stonework`, and `woodwork`, and assigned to roles such as `assistant`, `mason`, and `woodworker` by means of obligations and permissions (not shown in Figure 2(b)).

Lastly, the complete specification is used to organize the joint work of the human player with four artificial agents in building a house, as illustrated in Figure 2(a). The agents are instantiated by the player and run on the JADE platform. After creation, by activating o-behaviors, they connect to the MOISE+ run-time environment, assume roles, and commit to missions in coordination with each other. Furthermore, by means of w-behaviors, they actualize their decisions as character actions and interactions in the Minecraft virtual world.

*4.4. Discussion.* Far from a full-fledged system, we stress that the present implementation is meant as a prototype for showing the feasibility of the organizational gameplay proposed in this paper. In this sense, MOISE+ should be seen as a first approximation for a visual organizational language that can be used by game players. We note that MOISE+ was originally created as a general organizational language and infrastructure to be used by OC-MAS designers. It was not intended to be used by nontechnical people. Therefore, it is a rich language for organizational design that needs to be simplified and dressed up with an appealing visual notation to promote usability.

Besides MOISE+, other models [14, 22] can also be explored. Specifically, we have chosen MOISE+ due to its covering of structural and functional modeling concerns and the

availability of a well developed organizational infrastructure for implementing agent organizations. Both [14, 22] support the modeling of structural concerns in the form of groups and roles. But, different from MOISE+, they emphasize interaction protocols and/or interaction structures for describing the dynamics of organizations. In a first moment, taking into account usability, we have regarded the hierarchical plans of MOISE+ in the form of social schemes more appealing to the nontechnical user than interaction protocols based on scenes, states, states transitions, or interactions sequences.

Initially it was said that front-end, middle layer, and back-end can be instantiated in various ways. In the prototype, the front-end is based on MOISE+ language, the middle layer consists in JADE plus MOISE+ infrastructure, and the back-end is Minecraft game server. In this case, we have a multiplayer distributed configuration where all the components could run on different machines. This configuration also poses a clear distinction between the game engine (i.e., software for developing the virtual world and characters) and the agent organization infrastructure (i.e., software for developing agent players and agent organizations for the game). We stress that having this multiplayer distributed configuration with game engine distinct from the organizational infrastructure is not essential to our proposal. We think, and expect, that game engines could be extended to integrate the development of games based on agents organizations and that single person games can also be approached organizationally.

Regarding the agent players, we have designed fully organizational agents, or agents that guide their actions and interactions solely by organizational specifications. We also think that smarter agents could be created to balance not only reactivity and proactiveness, but also individual and organizational goals.

## 5. Related Work

A major challenge faced by game designers is how to incorporate high-level concerns such as flexible and/or social behaviors into the gaming experience. Usually, NPCs are programmed to react to environmental or other character conditions, instead of proactively pursuing explicit goals, be it alone or in collaboration with other characters. Hence, in most cases, their actions do not look natural from the human player perception.

In an attempt to overcome this limitation some research work have proposed the use of agent centered approaches to control characters. One of the first attempts to connect intelligent agents to games was Gamebots [23]. It provided an infrastructure to allow the connection of any agent platform and the Unreal Tournament game. Gamebots was later used as the base for Pogamut [24], a Java middleware that enables controlling virtual agents (NPCs) in multiple environments provided by game engines. Another noteworthy agent centered work is the GOAP architecture [25]. GOAP is a STRIPS-like planning architecture specifically designed for real-time control of autonomous character behavior in games. It is used in several commercial games such as F.E.A.R and Deus Ex: Human Revolution.

Regarding the use of organization centered MAS in game design, two relevant works are [26] and cOncienS [27]. In [26], the authors propose the use of agent organizations to control the coordination and adaptation in the context of serious games. Their aim is to adapt the learning experience to the skill level of each trainee, while maintaining the game believable and immersive. cOncienS [27] is a research work based on a larger project called ALIVE which defines an organizational framework to build service-oriented systems. It presents a set of tools for game developers to model scenarios using social structures. Specifically, it provides a programming interface for coupling agents to game engines, a tool for creating organization ontologies (specifications of agent organizations), and elements to describe character behaviors based on norms, roles, and their enactment.

In comparison, our proposal differs from these earlier works in a main aspect. Our research focuses on the use of *organization technology as gaming interface or gameplay*. That is, we propose the use of agent organizations not only as a design and implementation technology for more realistic and challenging NPCs, but also as a form of interaction between human players and the virtual world, a form of interaction that enables the user to explicitly perceive and reason about social/collective challenges rather than focusing only on individual skills. In this sense, we see the above agent centered and the organization approaches as complementary to ours (similarly to what happens with the traditional gameplay). In particular, we note that [26, 27] have explored the OperA [22] model which focus on dialogical structures and interactions protocols for scripting the dynamical aspects of agent organizations. Here, we have proposed the use of MOISE+ [17] which offers an alternative view based on goal decomposition tree for expressing collective behavior.

## 6. Conclusions

In this paper we have argued for an organizational interface for playing computer games as a enhancement to the existing gameplay. This *organizational gameplay*, as we call it, is built upon agent technology, more specifically borrowing elements from organization centered multiagent systems. The central idea is to enable game players to command more than one character by defining specifications for the structure and behavior of character organizations. At runtime, artificial agents are instantiated to control characters within such organizations on behalf of the player. To turn the idea into practice, we have proposed an abstract architecture comprising three components or layers and implemented a proof of concept prototype around Minecraft, JADE, and MOISE+.

The major benefit of the proposed organizational gameplay is to offer the player a novel perspective of the collective challenges faced while playing a game. In first person shooter games, for example, instead of controlling one character and implicitly assuming collaboration from the NPCs forming a group, the player actually could see and modify how the NPCs are supposed to behave as a group. Or, in real-time strategy games, instead of commanding a group of characters

by sending them frequent microinstructions, the player could plan ahead several strategies for different scenarios. Then, these strategies would be followed autonomously by the characters without the need of constant intervention by the player. In case of things going wrong, the user could switch to the traditional reactive mode of collective action. In sum, we believe that the importance of our proposal lies in providing a framework with which game designers can delve into a little explored area in computer games.

More than showing the technical feasibility of the proposal by using current technology, the implemented prototype has pointed us to further research work. First and foremost, we are investigating aspects of expressiveness versus usability of visual languages for organizational specification. On the one side, it is important to know what kinds of game challenges can be approached by explicit organizations; on the other, we want to know the complexity, from the point of view of the player, of dealing with organizational specifications. Regarding implementability, we noted that there should be an individual actions repertoire from which to compose collective goal decomposition trees. Presently, *behavior trees* (BT) are widely used to script the actions and interaction of game characters [28]. Based on these, we are investigating an agent architecture that could interpret BT, and if BT can also be used to express social schemes in the context of agent organizations.

Finally, as future work we plan to devise more effective ways to integrate our ideas with existing game engines. As we have pointed out in Section 1, we perceive a major opportunity to turn the gameplay into a more encompassing learning experience. In this regard, we also plan to conduct controlled experiments to assess the effectiveness of our proposal in teaching the players lessons of coordination, cooperation, and/or organization.

## Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

## Acknowledgments

The authors would like to thank UFMA and FAPEMA (Grants PPP-01368/11, BM-00457/12, BIC-03153/13, and BM-01245/14) for supporting this work.

## References

[1] R. Koster, *A Theory of Fun for Game Design*, O'Reilly Media, 2nd edition, 2013.

[2] G. Weiss, Ed., *Multiagent Systems*, The MIT Press, Cambridge, Mass, USA, 2nd edition, 2013.

[3] M. Luck, P. McBurney, and C. Preist, "A manifesto for agent technology: towards next generation computing," *Autonomous Agents and Multi-Agent Systems*, vol. 9, no. 3, pp. 203–252, 2004.

[4] M. Wooldridge, "Intelligent agents," in *Multiagent Systems*, G. Weiss, Ed., chapter 1, pp. 3–50, The MIT Press, 2013.

[5] F. Dignum, "Agents for games and simulations," *Autonomous Agents and Multi-Agent Systems*, vol. 24, no. 2, pp. 217–220, 2012.

[6] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with JaCaMo," *Science of Computer Programming*, vol. 78, no. 6, pp. 747–761, 2013.

[7] V. Dignum and J. Padget, "Multiagent organizations," in *Multiagent Systems*, G. Weiss, Ed., chapter 2, pp. 51–98, The MIT Press, Boston, Mass, USA, 2013.

[8] V. Dignum, Ed., *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, Information Science Reference, Hershey, 2009.

[9] B. Chaib-Draa, R. Mandiau, and P. Millot, "Distributed artificial intelligence, an annotated bibliography," *SIGART Bulletin*, vol. 3, no. 3, pp. 20–37, 1992.

[10] F. Zambonelli and A. Omicini, "Challenges and research directions in agent-oriented software engineering," *Autonomous Agents and Multi-Agent Systems*, vol. 9, no. 3, pp. 253–283, 2004.

[11] F. Zambonelli and H. V. D. Parunak, "Towards a paradigm change in computer science and software engineering: a synthesis," *The Knowledge Engineering Review*, vol. 18, no. 4, pp. 329–342, 2004.

[12] H. Hexmoor, C. Castelfranchi, and R. Falcone, *Agent Autonomy*, Springer, Boston, Mass, USA, 2003.

[13] C. Castelfranchi, "Engineering social order," in *Engineering Societies in the Agents World: First International Workshop, ESAW 2000, Berlin, Germany*, A. Omicini, R. Tolksdorf, and F. Zambonelli, Eds., vol. 1972 of *Lecture Notes in Computer Science*, pp. 1–18, Springer, Berlin, Germany, 2000, Revised Papers.

[14] J. Ferber, O. Gutknecht, and F. Michel, "From agents to organizations: an organizational view of multi-agent systems," in *Agent-Oriented Software Engineering IV: 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003, Revised Papers*, P. Giorgini, J. P. Müller, and J. Odell, Eds., vol. 2935 of *Lecture Notes in Computer Science*, pp. 214–230, Springer, Berlin, Germany, 2004.

[15] W. R. Scott, *Organizations: Rational, Natural and Open Systems*, Prentice Hall, Upper Saddle River, NJ, USA, 4th edition, 1998.

[16] E. Argente, O. Boissier, S. Esparcia, J. Grmer, K. Kirikal, and K. Taveter, "Describing agent organisations," in *Agreement Technologies*, S. Ossowski, Ed., pp. 253–275, Springer, Dordrecht, The Netherlands, 2013.

[17] J. F. Hubner, J. S. Sichman, and O. Boissier, "Developing organized multiagent systems using the MOISE+ model: programming issues at the system and agent levels," *International Journal of Agent-Oriented Software Engineering*, vol. 1, no. 3-4, p. 370, 2007.

[18] L. Coutinho, J. Sichman, and O. Boissier, "Modelling dimensions for agent organizations," in *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, V. Dignum, Ed., chapter 2, pp. 18–50, IGI Global, Hershey, Pa, USA, 2009.

[19] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-agent Systems with JADE*, John Wiley & Sons, Chichester, UK, 2007.

[20] R. H. Bordini, J. F. Hbner, and M. Wooldridge, *Programming Multi-Agent Systems in Agent Speak Using Jason*, John Wiley & Sons, 2007.

[21] J. Gregory, *Game Engine Architecture*, A K Peters, Wellesley, Mass, USA, 2009.

[22] V. Dignum, *A model for organizational interaction: based on agents, founded in logic [Ph.D. thesis]*, Utrecht University, 2004.

[23] G. A. Kaminka, M. M. Veloso, S. Schaffer et al., "GameBots: a flexible test bed for multiagent team research," *Communications of the ACM*, vol. 45, no. 1, pp. 43–45, 2002.

[24] J. Gemrot, R. Kadlec, M. Bída et al., "Pogamut 3 can assist developers in building ai (not only) for their videogame agents," in *Agents for Games and Simulations*, F. Dignum, J. Bradshaw, B. Silverman, and W. van Doesburg, Eds., vol. 5920 of *Lecture Notes in Computer Science*, pp. 1–15, Springer, Berlin, Germany, 2009.

[25] J. Orkin, "Agent architecture considerations for real-time planning in games," in *Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '05)*, R. M. Young and J. E. Laird, Eds., pp. 105–110, AAAI Press, 2005.

[26] J. Westra, F. Dignum, and V. Dignum, "Scalable adaptive serious games using agent organizations," in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '11)*, vol. 3, pp. 1291–1292, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, USA, May 2011.

[27] S. Alvarez-Napagao, I. Gómez-Sebastià, J. Vázquez-Salceda, and F. Koch, "cOncienS: organizational awareness in real-time strategy games," in *Proceedings of the Conference on Artificial Intelligence Research and Development: 13th International Conference of the Catalan Association for Artificial Intelligence (CCIA '10)*, vol. 210 of *Frontiers in Artificial Intelligence and Applications*, pp. 69–78, IOS Press, Amsterdam, The Netherlands, 2010.

[28] I. Millington and J. Funge, *Artificial Intelligence for Games*, Morgan Kaufmann, San Francisco, Calif, USA, 2nd edition, 2009.