Game Engines to Model MAS: A Research Roadmap

Stefano Mariani
DISI, ALMA MATER STUDIORUM–Università di Bologna
via Sacchi 3, 47521 Cesena, Italy
Email: s.mariani@unibo.it

Andrea Omicini
DISI, ALMA MATER STUDIORUM–Università di Bologna
via Sacchi 3, 47521 Cesena, Italy
Email: andrea.omicini@unibo.it

***Abstract*—Game engines are gaining increasing popularity in various computational research areas, and in particular in the context of multi-agent systems (MAS)—for instance, to render augmented reality environments, improve immersive simulation infrastructures, and so on. Existing examples of successful integration between game engines and MAS still focus on specific technology-level goals, rather than on shaping a general-purpose game-based agent-oriented infrastructure. In this roadmap paper, we point out the conceptual issues to be faced while attempting to exploit game engines as agent-oriented infrastructures, and outline a possible research roadmap to follow, backed up by some early experiments involving the Unity3D engine.**

I. RESEARCH LANDSCAPE

As a preparatory background for the remainder of the paper, in this section we briefly recall agent-oriented abstractions (Subsection I-A) and overview early attempts to exploit game engines within multi-agent systems (Subsection I-B), motivating the need for further advancing their pioneer experience (Subsection I-C).

A. Agents & MAS

Agent-oriented models and technologies represent the richest sources of abstractions and mechanisms for the engineering of complex software systems—in particular, those demanding for advanced features such as distribution, interoperability, intelligence, mobility, and autonomy [1]. Multi-agent systems (MAS) in particular provide a well-established technological framework for all those scenarios mandating for decentralisation, fault-tolerance, adaptiveness, situatedness, and self-organisation. As its basic design abstractions, any MAS features *agents* as the proactive components of the system, the *environment* as the (mainly) reactive, either virtual or physical context where the agents are situated, and agent *societies* as a way to capture and possibly govern agents relationships of any sort [2].

More in detail, agents are computational entities whose defining feature is *autonomy* [3]. Agents model activities within a MAS, expressed in terms of their actions along with their motivations—namely, the *goals* and intentions that prompt and set the agent's course of actions. Agent societies represent then the groups where MAS collective behaviours are coordinated towards the achievement of the overall system goals. *Coordination models* and *languages* are then the most suitable tools to tackle complexity in MAS [4], as they are explicitly meant to supply the abstractions that "glue" agents together [5], [6] by governing agent interaction [7].

Besides agents and societies, environment is an essential abstraction for MAS modelling and engineering [2], to be suitably represented, and related to agents. On the one side, the notion of environment captures the unpredictability of the MAS context, by modelling the external resources and features that are relevant for the MAS, along with their evolution over time. On the other side, it makes it possible to model the resources, tools, services that agents and MAS need to carry on their own activities. Along with the notion of *situated action* – as the realisation that coordinated, social, intelligent action arises from strict interaction with the environment, rather than from rational practical reasoning [8] – this leads to the requirement of *situatedness* for agents and MAS, often translated into the need of being sensitive to *environment change* [9].

B. Game Engines

Game Engines (GE) are increasingly popular in many different areas of computer science. In particular, they are mostly used in order to implement two MAS abstractions, that is, agents and MAS environment—typically, in specific domains, aimed at achieving some specific goals. For instance,

- QuizMASter [10] focusses on the agent abstraction by linking MAS agents to game engines characters, in the context of educational learning
- CIGA [11] considers both agents and environment modelling, for general-purpose virtual agents in virtual environments
- GameBots [12] focusses on the agent abstraction, but still considers environment while providing a development framework and runtime for multi-agent systems testing in virtual environments
- UTSAF [13] focusses on environment modelling in the context of distributed simulations in the military domain[1]

Although they clearly represent examples of (partially) successful integration of MAS within GE, the aforementioned works share a few shortcomings w.r.t. the goal we pursue in this paper:

[1] Agents are considered, but only as an integration means between different simulation platforms, not in the context of the GE exploited for simulation rendering.

- with the only exception of CIGA (which recognises the conceptual gap between MAS and GE, and proposes solutions to deal with it—although on the technological level), the only layer taken into account while pursuing integration is the technological one—no model, no architecture, no language
- integration is mostly domain- and goal-specific—as it happens for instance in QuizMASter and UTSAF, and even in GameBots to some extent[2]
- whereas most approaches provide programmers with some abstractions to deal with agents and environment, no attention is given to social abstractions

C. Motivation

Besides the lack of completeness and generality just highlighted, and the increasing interest in the topic, a few other considerations motivate worthiness of our research efforts.

First of all, there is huge gap in the technological advancement GE have reached w.r.t. the technological level of agent-oriented infrastructures born within the academic community. This should not surprise anyone: the gaming scene may rely on a billionaire industry and on millions of developers and testers (besides gamers), which are well paid to push stability, performance, usability of their products to unprecedented and incomparable levels of quality. Therefore, it is worth to consider the possibility of taking advantage of such a finely-optimised products for improving the overall quality of MAS technologies.

Second, and dually, there is a huge gap in the conceptual and design abstractions GE provides to developers w.r.t. far richer abstractions agent-oriented software engineering provides. All the GE we considered in this paper provide really low level abstractions, especially on the agent side, where, for instance, programming cyclic behaviour amounts to write coroutines spanning multiple rendering stages.

Third, integrating MAS with GE may provide novel solutions to deal with the typical issues of augmented reality scenarios, such as those targeted by the mirror worlds model [14], in which coordination is requested to be space-aware and spatially situated in a physical environment.

For the above reasons, in this paper we provide the foundation for a research roadmap towards a well-founded integration of GE and MAS, with particular emphasis on how GE could be exploited so as to model MAS constituent abstractions not just as a solution to a rendering problem, but as they provide rather rich features and stimulating opportunities, in general. Accordingly, Section II tries to map MAS abstractions upon those provided by two exemplary and widespread GE—Unity3D (Subsection II-A) and Unreal Engine (Subsection II-B); Section III discusses the general issue of exploiting GE to model MAS given the aforementioned analysis, providing a first working prototype of a tuple-based coordinated MAS [15] implemented in Unity3D as a proof-of-concept; finally Section IV provides final remarks along with a research roadmap for organising exploration of such a novel research line.

[2]It focusses on rapid prototyping and testing MAS in virtual worlds.

II. Agents, Environment, and Societies in Game Engines

Game Engines (GE) are frameworks for supporting design and development of games. Modern GE are all-around frameworks geared toward every aspect of game design and development, such as 2D/3D rendering of game scenes, physics engines for environmental dynamics (movements, particles dynamics, collision detection, obstacle avoidance, etc.), sounds, behavioural scripting, characters' artificial intelligence, and much more.

As meaningful examples well representing the full range of available platforms, in this section we examine two of the most popular and used GE – Unity3D and Unreal Engine – with the aim of:

- detecting those abstractions and mechanisms most likely to have a counterpart in MAS, or at least those which seem to provide some support in re-formulating MAS missing abstractions
- highlighting opportunities for closing conceptual/technical gaps hindering integration of the two worlds

A. Unity3D

Unity3D[3] is developed by Unity Technologies, and (as of version 5.3.4) it features a few abstractions worth to be mentioned here:

- the *game object*, which actually is the only *first-class citizen* in Unity3D world, being everything a scene contains a game object: a human player, a Non-Player Character (NPC), an environmental item, everything
- the *script*, which is a piece of code defining the *behaviour* to be attached to a game object; scripts are executed by the *unique* Unity3D game loop, which *sequentially* executes *once* each script at each game frame rendered—no concurrency, *all* the scripts must be executed within *each frame* rendering step
- the *co-routine*, which acts as a sort of workaround to sequentiality imposed to scripts, by enabling developers to *partition* a computation and *distribute* its pieces over *multiple frame rendering steps*, suspending and resuming execution at precise points within the code through explicit API calls

Looking at MAS, the abstraction gap is quite wide, being Unity3D totally missing an agenthood abstraction as well as dedicated abstractions to handle sociality.

The only MAS abstraction somehow represented and directly supported is that of *environment*, through game objects. Indeed, in some sense, its support to environment modelling, control, and interaction is far superior w.r.t. the average MAS technology – e.g. JADE[16], *Jason* [17], RETSINA [18], TuCSoN [19], CArtAgO [20] –, since Unity3D is capable of directly supporting many forms of *agent-environment interaction*, such as shaping movement pathways, handling obstacle avoidance and collision detection, and the like.

[3]http://unity3d.com

As far as agenthood is concerned, no first-class abstraction is provided by Unity3D. However, there is still an opportunity for bridging the gap, although mostly through a workaround: the combination of a game object with an attached behavioural script (and possibly a co-routine too) is the best we can ask for to Unity3D. However, most of the behavioural logic would have to be implemented by the developer with little or even no support from Unity3D.

To conclude analysis of Unity3D, also a societal abstraction is missing among first-class ones. However, the possibility to leverage some sort of *message passing*[4] among game objects, in various forms such as unicast based on objects unique names, multicast based on tags attributed to arbitrary group of objects, and broadcast by generally referring to the scene, makes it possible to implement message-based interaction—and coordination, to some extent (see Subsection III-A).

Summing up, both agenthood and sociality present a quite large abstraction gap to fill in order to conceptually frame an integration effort, whereas on the environment side, although the gap is less apparent, mapping is still far from being perfect. On a more technical level, there are opportunities for exploiting workarounds, but the implementation effort to reconstruct even a primitive support to agenthood is expected to be huge, the same holding for more complex forms of interaction, such as asynchronous conversations and full-fledged protocols.

B. Unreal Engine

Unreal Engine[5] is developed by Epic Games, and (as of version 4) it features a few abstractions worth to be described for the purpose of the paper:

- the *game object*, similar to Unity3D game objects, except they are not the only first-class abstractions—see below
- the *actor*, which is any game component which can be rendered, and whose behaviour is enacted by a controller, either interfacing a human player or an artificial intelligence (bot); *characters* are a special kind of actor with humanoid resemblance and capabilities (e.g. walking)
- the *blueprint*, similar to Unity3D script, being more or less the code[6] specifying the behaviour of an actor
- the *direct blueprint communication*, which enables blueprints to communicate one-to-one; also, the *event* is a game-related happening (e.g. level started, damage taken, shots fired, etc.) which may trigger execution of blueprints code; finally, the *event dispatcher* plays the role of producer in a publish/subscribe-like communication architecture, where blueprints always play the role of consumers

As far as the MAS model is concerned, the abstraction gap is still considerable—yet possibly smaller, if compared to Unity3D.

[4]Actually, sending a message to an entity requires to specify which method the receiver is expected to execute to handle reception.

[5]http://www.unrealengine.com

[6]Actually, blueprints are programmed visually, by wiring functions, data, etc. in a graphical editor.

In fact, agenthood can be reconstructed based on actors and blueprints, whereas social interaction may be engineered on top of event dispatchers and direct blueprint communications—although still with considerable conceptual and technical effort. Finally, on the environment side, the situation is almost identical to the one described for Unity3D.

III. GE & MAS: Towards the Integration

Based on the analysis reported in Section II, integration between GE and MAS is certainly interesting, likely useful, and seemingly possible—although with considerable effort according to the state of art.

It is worth to note here that besides trying to reconstruct MAS abstractions by (ab)using GE features, another path toward integration is available, promoted by, e.g., literature on *mirror worlds* [14], that is, virtual representations of the real world—which in this case might be virtual too. In fact, it is possible to imagine MAS agents exploiting GE to represent the physical/virtual environment they are immersed in (*situated*) so as to delegate to the GE handling of, e.g., movement, environment-mediated interaction, discovery, and the like. The social dimension may be kept within the MAS world if direct communication between agents is needed, or expanded to the GE virtual world if communication mediated by the environment is what MAS designers are looking for. Nevertheless, the virtual world managed by the GE may influence social interactions, e.g., by either facilitating or hindering discovery.

In the following section, we report on an early experiment conducted to verify feasibility of the aforementioned approach.

A. A Case Study

We test the extent to which workarounds can be exploited to reconstruct some (approximated) MAS notions – agenthood, social interaction, and environment mediation – in Unity3D by implementing the most well-known scenario involving all the aforementioned facets of a MAS: the Dining Philosophers (DP) coordination problem, tackled in a shared-space setting.

In the DP scenario, depicted in Figure 1, five philosophers are sharing a table with a big spaghetti bowl, five smaller bowls, five chopsticks, and five chairs (one for each philosopher)—the scenario can be easily extended to an arbitrary number of philosophers. Thus, there are agents – the philosophers – and an environment with a few shared resources—the chopsticks and the chairs. On the society side, philosophers interact and coordinate by exploiting environment mediation: in fact, we choose to solve the dining philosophers coordination problem adopting a tuple-based approach [15]. Accordingly, the table plays the role of the tuplespace, chopsticks of shared tuples, and chairs decouple and mediate interaction, while enabling situated coordination.

First of all, the tuple space is implemented by decorating the table game object with a few properties: a `tupleSet` (list of strings) representing the tuple storage medium, a `inputQueue` tracking requests for tuples (Unity3D messages) as soon as they arrive, and a `pendingQueue` tracking requests yet unsatisfied. Three request types are supported: `out`, `in`, `rd`. Operationally, at every frame rendering step, the table script does the following:

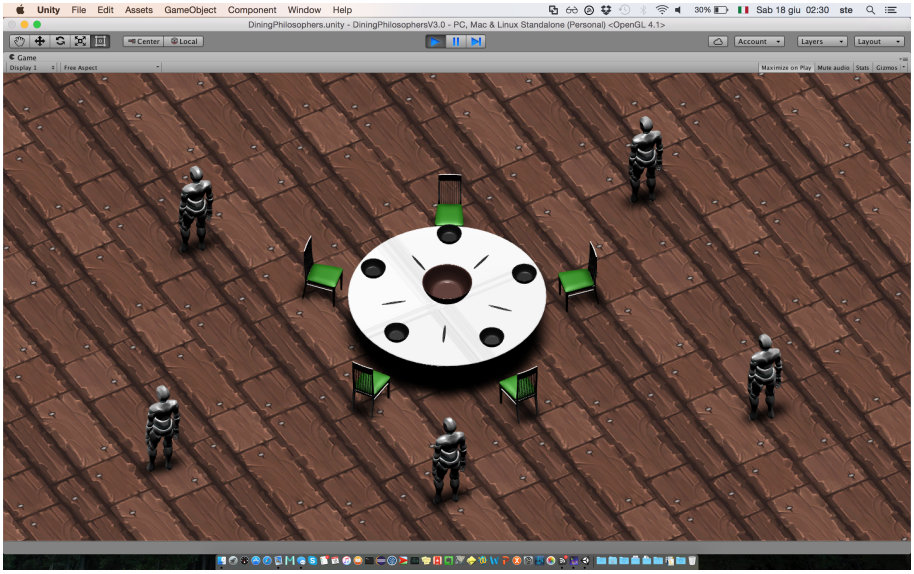

Figure 1. Dining philosophers in Unity3D. Grey philosophers are thinking. Green chairs are free.

1) the `pendingQueue` is considered first, and the request type is checked
   - in case of an `out`, the `pendingQueue` is checked looking for pending requests having a matching template
     - in case there is a pending request, it is served and removed from the queue
     - in case there is not, the tuple is inserted in the `tupleSet`
   - in case of an `in`, if it is satisfiable, that is, a tuple matching the given template (simple regular expressions on strings) exists, the tuple is removed from the `tupleSet` and given to the requestor—whose reference is dynamically retrieved
   - in case of a `rd`, if it is satisfiable the tuple is not removed and given to the requestor
2) the `inputQueue` is considered then, and the request type is checked as above
3) in case requests cannot be satisfied, they are either removed from the `inputQueue` and put into the `pendingQueue`, or kept in the latter

Agent game objects are implemented to request the three coordination primitives provided by the table synchronously, exploiting Unity3D co-routines: for every request sent, the agent waits for a reply, suspending execution (actually, delaying to next frame rendering step) until it arrives.

Chairs work as decouplers of interaction as well as situatedness enablers, by letting philosophers dynamically acquire the right chopsticks based on their position at the table. In fact, philosopher agents asks for chopsticks to the chair they are currently sit on, rather than directly to the table, and do not explicitly refer chopsticks: not by name, neither by address, nor by any other means. Each chair dynamically knows where it is w.r.t. the table and therein placed chopsticks thanks to raytracing, enabling game objects to detect nearby objects. Then, it is the chair that asks the table for the right pair of chopsticks on behalf of the philosopher agent that is currently sitting on it.

The scenario works as follows:

1) each philosopher thinks until it gets hungry
2) when this happens, it looks for a free chair to sit—exploiting raytracing
3) when a free chair is found, the philosopher goes there – movement pathway and collision avoidance are a "free lunch" when using GE – and sits, waiting to acquire the chopsticks required according to the position of the chair w.r.t. the table
4) when a chair gets occupied, it acquires information about which chopsticks should be handed over to the hungry philosopher – through raytracing – and performs the corresponding requests to the table—on behalf of the agent
   - in case the requests are satisfied, the philosopher starts eating (Figure 2)—and leaves the chair to think again when done, thus restarting its thinking-eating loop
   - in case at least one request cannot be satisfied, the philosopher waits as described above (Figure 3)

All interactions happen through Unity3D messaging facilities.

IV. Discussion & Roadmap

The implementation effort described in Subsection III-A, although rather successful on the technical side, raises many conceptual issues regarding MAS and GE integration in the same infrastructure.

First of all, agent autonomy is all but trivial to preserve: regardless of the approach taken to marry MAS and GE – either the reconstruction one or the mirror worlds one – autonomy is likely to be the toughest issue to tackle. For GE very own nature, the flow of control should be in their hands no matter what, because fluidity of the rendering process is the foremost concern. This is an obvious clash with the definition of autonomy as encapsulation of the flow of control and of the

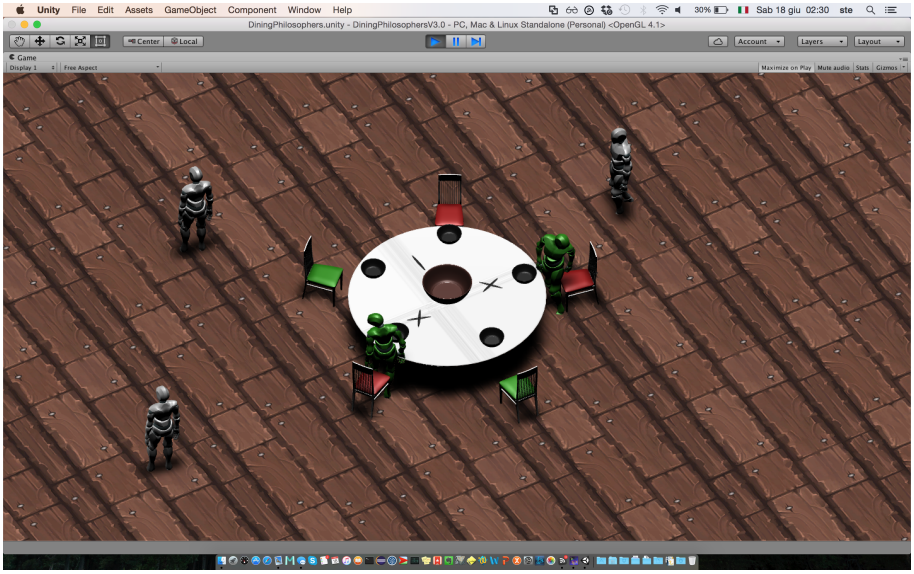

Figure 2. Eating philosophers. Green philosophers are those eating. Red chairs are occupied (reserved) by philosophers willing to eat.

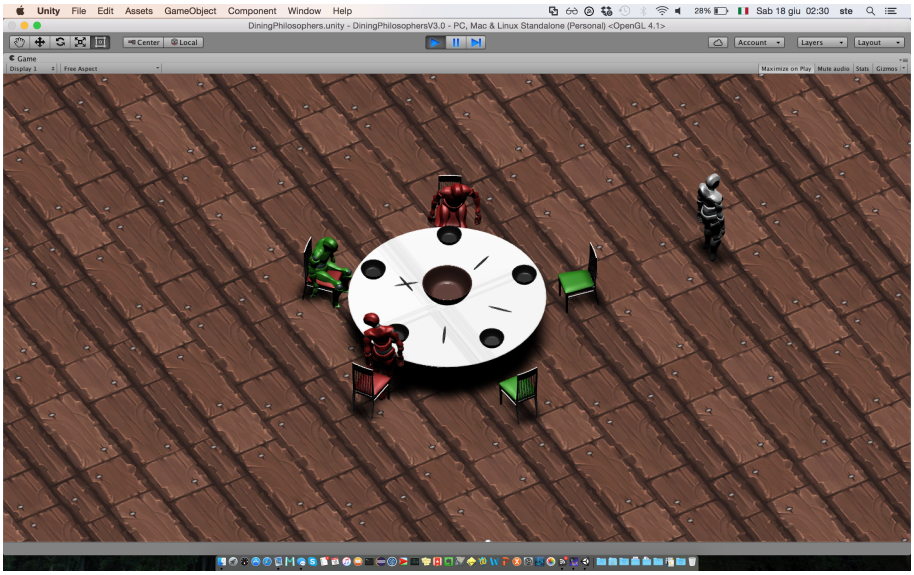

Figure 3. Waiting philosophers. Red philosophers are willing to eat, but obliged to wait until chopsticks become available.

criteria to handle it. However, co-routines in Unity3D and the actor abstraction in Unreal Engine may be exploited to build some notion of autonomy, and/or for linking virtual agents in a mirror world with their MAS counterparts while preserving their autonomy.

As far as situatedness is concerned, thus focussing on the environment abstraction, things get better: being environmental resources and agent-environment interaction first-class citizens in any GE, any MAS willing to exploit GE facilities is likely to benefit of some free lunch—e.g. movement pathways computation, objects discovery, collision avoidance.

Finally, when focussing on the social side, issues differ depending the kind of interaction: for direct interactions among agents, support provided by GE is limited to basic message-passing facilities; for environment-mediated interactions, support is a lot better thanks to situatedness-related features such as discovery, routing, etc.

Summing up, from the early conceptual speculations and practical experimentations undertaken in this work, we can guess that integration is likely to happen, at least at first, mainly through environmental abstractions, since they are the most well represented in the GE world. Accordingly, we conclude the paper by outlining the research roadmap we think is the more likely to lead to some successful results.

First of all, the *mirror worlds* approach seems the most appropriate, given that the reconstruction approach is hindered by the constraints on the flow of control imposed by state-of-art-GE technologies.

Then, *environmental abstractions* as provided by MAS and GE should be carefully analysed to seek for mapping opportunities, with the aim of drawing correspondences between

the GE-based virtual representation of the MAS world—which could be virtual, physical, hybrid. An interesting path to follow in this sense is represented by, e.g., the *artefact* abstraction [20] as defined in the A&A meta-model for MAS [3].

Once that the environment layer of the envisioned GE-based MAS is settled, agenthood could be (re)shaped around this, by exploiting GE capabilities regarding *situated interaction*. For instance, whereas the goal-directed/oriented behaviour of agents could be still programmed on top of the more expressive mechanisms provided by traditional MAS paradigms, such as BDI reasoning, it could also take advantage of GE features for, e.g., *practical reasoning* and *situated planning*, relying on GE virtual world representation to, e.g., estimate the effects of actions.

Sociality too can be (re)shaped around the environmental layer: for instance, whereas direct communication between agents may be still allowed and based on traditional MAS mechanisms, accessory facilities may be delegated to the GE part of the system, such as discovery of recipients based on *spatial proximity*.

This way, a GE-based MAS infrastructure could in principle be designed, by suitably integrating the different mechanisms and paradigms brought by GE and MAS around environmental abstractions, while avoiding to abuse either technology so as to carrying out activities and pursuing goals it was not meant to deal with—such as, for instance, attempting to reconstruct some notion of autonomy for GE agents.

ACKNOWLEDGEMENTS

We would like to thank Mattia Cerbara and Nicola Poli for their analysis of Unity3D and Unreal Engine, and for the prototype implementation of the case study. Also, our gratitude goes to the reviewers, stimulating the discussion in Section IV with their stimulating questions and thoughtful remarks.

REFERENCES

[1] 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, Nov. 2004, Special Issue: Challenges for Agent-Based Computing. [Online]. Available: http://link.springer.com/10.1023/B:AGNT.0000038028.66672.1e

[2] D. Weyns, A. Omicini, and J. J. Odell, "Environment as a first-class abstraction in multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 14, no. 1, pp. 5–30, Feb. 2007, Special Issue on Environments for Multi-agent Systems. [Online]. Available: http://link.springer.com/10.1007/s10458-006-0012-0

[3] A. Omicini, A. Ricci, and M. Viroli, "Artifacts in the A&A meta-model for multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 3, pp. 432–456, Dec. 2008, special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems. [Online]. Available: http://link.springer.com/10.1007/s10458-008-9053-x

[4] P. Ciancarini, A. Omicini, and F. Zambonelli, "Multiagent system engineering: The coordination viewpoint," in *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, ser. LNAI, N. R. Jennings and Y. Lespérance, Eds. Springer, 2000, vol. 1757, pp. 250–259. [Online]. Available: http://link.springer.com/10.1007/10719619_19

[5] D. Gelernter and N. Carriero, "Coordination languages and their significance," *Communications of the ACM*, vol. 35, no. 2, pp. 97–107, 1992. [Online]. Available: http://dl.acm.org/citation.cfm?doid=129635

[6] P. Ciancarini, "Coordination models and languages as software integrators," *ACM Computing Surveys*, vol. 28, no. 2, pp. 300–302, Jun. 1996. [Online]. Available: http://dl.acm.org/citation.cfm?doid=234732

[7] P. Wegner, "Coordination as constrained interaction," in *Coordination Languages and Models*, ser. Lecture Notes in Computer Science, P. Ciancarini and C. Hankin, Eds. Springer, 1996, vol. 1061, pp. 28–33. [Online]. Available: http://link.springer.com/10.1007/3-540-61052-9_37

[8] L. A. Suchman, "Situated actions," in *Plans and Situated Actions: The Problem of Human-Machine Communication*. New York, NYU, USA: Cambridge University Press, 1987, ch. 4, pp. 49–67.

[9] J. Ferber and J.-P. Müller, "Influences and reaction: A model of situated multiagent systems," in *2nd International Conference on Multi-Agent Systems (ICMAS-96)*, M. Tokoro, Ed. Tokio, Japan: AAAI Press, Dec. 1996, pp. 72–79.

[10] J. Blair and F. Lin, "An approach for integrating 3D virtual worlds with multiagent systems," in *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, Mar. 2011, pp. 580–585. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5763564

[11] J. van Oijen, L. Vanhée, and F. Dignum, *CIGA: A Middleware for Intelligent Agents in Virtual Environments*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 22–37. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32326-3_2

[12] G. A. Kaminka, M. M. Veloso, S. Schaffer, C. Sollitto, R. Adobbati, A. N. Marshall, A. Scholer, and S. Tejada, "GameBots: A flexible test bed for multiagent team research," *Communications of the ACM*, vol. 45, no. 1, pp. 43–45, Jan. 2002. [Online]. Available: http://dl.acm.org/citation.cfm?doid=502293

[13] P. Prasithsangaree, J. Manojlovich, S. Hughes, and M. Lewis, "UTSAF: A multi-agent-based software bridge for interoperability between distributed military and commercial gaming simulation," *Simulation*, vol. 80, no. 12, pp. 647–, 2004. [Online]. Available: http://sim.sagepub.com/content/80/12/647

[14] A. Ricci, A. Croatti, P. Brunetti, and M. Viroli, "Programming mirror worlds: An agent-oriented programming perspective," in *Engineering Multi-Agent Systems*, ser. Lecture Notes in Computer Science, M. Baldoni, L. Baresi, and M. Dastani, Eds. Springer, 2015, vol. 9318, pp. 191–211, 3rd International Workshop, EMAS 2015, Istanbul, Turkey, May 5, 2015, Revised, Selected, and Invited Papers. [Online]. Available: http://link.springer.com/10.1007/978-3-319-26184-3_11

[15] D. Rossi, G. Cabri, and E. Denti, "Tuple-based technologies for coordination," in *Coordination of Internet Agents: Models, Technologies, and Applications*, A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, Eds. Springer, Jan. 2001, ch. 4, pp. 83–109. [Online]. Available: http://link.springer.com/10.1007/978-3-662-04401-8_4

[16] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Wiley, Feb. 2007. [Online]. Available: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470057475.html

[17] R. H. Bordini, J. F. Hübner, and M. J. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, Ltd, Oct. 2007. [Online]. Available: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470029005.html

[18] K. Sycara, M. Paolucci, M. Van Velsen, and J. Giampapa, "The RETSINA MAS infrastructure," *Autonomous Agents and Multi-Agent Systems*, vol. 7, no. 1, pp. 29–48, 2003. [Online]. Available: http://link.springer.com/10.1023/A:1024172719965

[19] A. Omicini and F. Zambonelli, "Coordination for Internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sep. 1999, special Issue: Coordination Mechanisms for Web Agents. [Online]. Available: http://link.springer.com/10.1023/A:1010060322135

[20] A. Ricci, M. Piunti, and M. Viroli, "Environment programming in multi-agent systems – an artifact-based perspective," *Autonomous Agents and Multi-Agent Systems*, vol. 23, no. 2, pp. 158–192, Sep. 2011, Special Issue: Multi-Agent Programming. [Online]. Available: http://link.springer.com/10.1007/s10458-010-9140-7