Models of Autonomy and Coordination: Integrating Subjective and Objective Approaches in Agent Development Frameworks

Stefano Mariani, Andrea Omicini, and Luca Sangiorgi

Abstract. Objective and subjective approaches to coordination constitute two complementary approaches, which, being both essential in MAS engineering, require to be suitably integrated. In this paper, we *(i)* observe that a successful integration depends on the *models of autonomy* and *coordination* promoted by agent technologies, *(ii)* suggest that ignoring the two models may hinder agent autonomy, *(iii)* provide an example of "autonomy-preserving" integration by discussing TuCSoN4JADE.

1 Autonomy and Coordination: Issues

Autonomy is a core notion for agents as social entities in a multiagent system (MAS) [8]. *Interdependency* between agent activities is the foundation for the need for *coordination* [4], which becomes an essential facet of MAS design, as relevant as individual agent design [5]. Coordinating a society of agents towards the achievement of a social goal necessarily influences agent course of action, potentially hindering their autonomy—in particular when adopting the *objective* approach to the design of MAS coordination [13].

Objective coordination [13] refers to the coordination approaches – typical of the Software Engineering research field – where coordination-related concerns are extracted from agents to be embodied within dedicated abstractions – e.g., *coordination artefacts* [14] – offering *coordination as a service* to agents [23]. Coordination abstractions steer agent societies towards the achievement of social goals by managing dependencies between agent activities, even despite individual agent goals. On the other hand, *subjective coordination* [13] represents the dual approach, as it is typically adopted in the (Distributed) Artificial Intelligence field [10]. There, coordination issues are directly tackled by individual agents themselves, determining

Stefano Mariani · Andrea Omicini · Luca Sangiorgi
ALMA MATER STUDIORUM–Università di Bologna
via Sacchi 3, 47521 Cesena, FC, Italy
e-mail: {s.mariani,andrea.omicini}@unibo.it,
luca.sangiorgi6@studio.unibo.it

their best course of action in the attempt to achieve their own goals—which typically requires the intelligence to perform practical reasoning. As the result of agent own deliberation activity, subjective coordination basically expresses agent autonomy.

Thus, objective and subjective coordination clearly constitute two *complementary* approaches, both essential in MAS design and development [19], hence requiring to be suitably integrated—as also witnessed by many results already achieved. In [19], Activity Theory was proposed as the conceptual framework reconciling the objective and subjective approach, whereas in [16] TuCSoN coordination infrastructure [17] and JADE agent development framework [2] were integrated by offering TuCSoN tuple-based coordination as a JADE service. In [20], the CArtAgO framework [21] was integrated with three different agent platforms – Jason [3], 2APL [6] and simpA [22] – so as to enable and promote artefact-based interaction of heterogeneous agents.

In this paper, we first observe that the successful integration of objective and subjective coordination strongly depends on the technology level, that is, on the abstractions and mechanisms actually promoted by the agent frameworks. In particular, when building a MAS, integration depends on the *model of autonomy* promoted by the specific agent platform, and by its relationship with the *model of coordination* implemented by the specific (objective) coordination framework. Then, we show that any integration effort not taking into account such two aspects is likely to hinder agent autonomy by (unintentionally) creating *artificial dependencies* between the individual and the social stances on coordination. Finally, we provide an example of effective integration of objective and subjective coordination by discussing TuCSoN4JADE[1], where the model of autonomy promoted by the JADE platform is seamlessly integrated with the model of coordination provided by the TuCSoN middleware.

2 Autonomy and Coordination: Models and Technologies

Given the centrality of autonomy in the definition of agents, any agent development framework is required to provide architectural solutions to enable and support agent autonomy. Either explicitly or implicitly, such architectures assume what we call a *model of autonomy*, that is, a model defining *(i)* how agents behave as individual (autonomous) entities, *(ii)* how they relate to each other as social entities, as well as *(iii)* how the two things coexist. In Subsection 2.1 we analyse the model of autonomy promoted by two well-known agent development frameworks: JADE [2] and Jason [3].

In a similar way, the architectural components offering coordination services in agent infrastructures adhere to a *model of coordination*, which defines the semantics of the admissible interactions between agents in a MAS, in particular, w.r.t. their effects on the agent's control flow—hence, on agent autonomy. In Subsection 2.2 we

[1] Available at `http://bitbucket.org/smariani/tucson/downloads`

analyse the model of coordination provided by two well-known agent infrastructures: TuCSoN [17] and CArtAgO [21].

2.1 Autonomy and Coordination in Agent Development Frameworks

Jade

Jade (Java Agent DEvelopment Framework) [2] is a Java-based framework and infrastructure to develop open, distributed agent-based applications in compliance with FIPA standard specifications for interoperable, intelligent, multi-agent systems. In Jade, autonomy of agents is supported by the *behaviour* mechanism, whereas their mutual interaction depends on the *Agent Communication Channel* (ACC).

A behaviour can be logically interpreted as "an activity to perform with the goal of accomplishing a task". Thus, different "courses of actions" of a Jade agent are encapsulated into distinct behaviours the agent executes simultaneously. Technically, Jade behaviours are Java objects, which are executed *pseudo-concurrently* within a single Java thread by a *non-preemptive, round-robin scheduler*. During Jade agent initialisation, behaviours are added to the *ready queue*, ready to be scheduled. Then, method `action()` of the first behaviour – containing the agent's "course of action" – is executed. "Behaviours switch" occurs only when such method returns; hence, meanwhile *no other behaviour can start execution*—"non-preemptive" scheduler. Behaviour removal from the ready queue occurs only when the `done()` method returns `true`; otherwise, the behaviour is re-scheduled at the end of the queue—"round-robin" scheduler. Notice method `action()` is executed *from the beginning every time*: there is no way to "stop-then-resume" a behaviour.

The ACC is the run-time facility in charge of *asynchronous message passing* among agents: each agent has its own mailbox, and is notified upon reception of any message. Jade agents can communicate via several methods, among which:

`receive()`	to asynchronously retrieve the first message
`blockingReceive()`	to perform a *synchronous* receive

According to the Jade Programmers Guide [1], some care should be taken in using method `blockingReceive()`: in fact, it suspends *the agent*, not only the calling behaviour. This semantics impacts the aforementioned third dimension of the model of autonomy: "how the two things coexist". In fact, resorting to a synchronous communication mechanism hinders autonomy of the caller agent, since all its other behaviours – not just the caller one – are suspended by the communication semantics. In order to preserve agents autonomy, the Jade Programmers Guide suggests adoption of the following programming pattern: call `receive()` instead, then call method `block()` – of the `Behaviour` class – if no message is found, so as to let Jade suspend only the calling behaviour. The ubiquity of such pattern in JADE code factually witnesses the relevance of the issue of understanding and suitably define the model of autonomy.

Summing up, JADE model of autonomy features *(i)* behaviours for individual tasks, *(ii)* asynchronous messages for subjective coordination, *(iii)* the "`block()`-then-resume" pattern to reconcile individual and social attitudes. Subsection 3.1 shows how any integration effort ignoring this semantics is bound to fail.

Jason

Jason [3] is both an agent language and an agent run-time system. As a language, it implements a dialect of AgentSpeak [18]; as a run-time system, it provides the infrastructure needed to execute a MAS. Although Jason is entirely programmed in Java, it features BDI agents, so a higher-level language (the Jason language) is used to program Jason agents using BDI abstractions. In Jason, autonomy of agents is supported by the Jason *plan/intention* execution machinery and the message passing facilities.

Like JADE behaviours, a Jason *plan* can be interpreted as a course of action to be performed to accomplish a task. Technically, a Jason plan differs considerably from JADE behaviours: *(i)* it is scheduled for execution as soon as a *triggering event* occurs, *(ii)* it is not directly executed "as is" (in general), but is instantiated as an *intention*, then executed, *(iii)* intentions are pseudo-concurrently executed *one action each*, according to a round-robin scheduler. Whereas in JADE the behaviour is the basic execution step, in Jason the same role is played by the single action, not by the plan/intention. Intentions may be *suspended* by the Jason reasoner, e.g. because the agent needs to wait for a message.

Jason agents can in fact exchange beliefs/plans/goals in the form of messages. Thus, subjective coordination is supported by these message passing facilities. In Jason, intentions are automatically suspended whenever they perform a "communication action" which cannot complete—to be resumed as soon as the action obtains its "completion feedback" (see [3], page 86). This preserves Jason agent autonomy similarly to behaviours in JADE: namely, by decoupling the control flow of a given "course of action" from the one of the agent undertaking them.

Summing up, Jason's model of autonomy features *(i)* plans/intentions for individual tasks, *(ii)* asynchronous message passing for subjective coordination, *(iii)* intention suspension mechanism to reconcile individual and social attitudes.

2.2 Autonomy and Coordination in Agent Infrastructures

TuCSoN

TuCSoN [17] is a Java-based, (logic) tuple-based coordination model and infrastructure for open, distributed MAS. It extends the LINDA model [7] by featuring

ReSpecT *tuple centres* [12] as its coordination artefacts [14], which are distributed over a network of TuCSoN nodes.

The TuCSoN architectural component that mostly explains its model of coordination is the *Agent Coordination Context* (ACC) [11]. ACCs are assigned to agents as they enter a TuCSoN-coordinated MAS to map coordination operations into events, *asynchronously* dispatching them to the coordination medium. Thus, ACCs are fundamental to guarantee and preserve agent autonomy: while the agent is free to choose and undertake its course of actions, its associated ACC takes care of communicating "coordination-related" events to TuCSoN—and of collecting results. In particular, ACCs enable separation of the *suspensive semantics* of a coordination operation from its *invocation semantics*. More precisely, the suspensive semantics implies that the operation itself is suspended if needed. Instead, the *synchronous* invocation semantics implies that *the agent, too* is suspended if the operation gets suspended.

To do so, every TuCSoN operation execution undergoes two steps:

invocation | the request to carry out a given coordination operation is sent to the TuCSoN tuple centre target of the operation

completion | the response to the coordination operation invoked is sent back to the requesting agent by the tuple centre

In other terms, any coordination operation in TuCSoN is *asynchronous by default*. Nevertheless, each of the TuCSoN coordination operations can be invoked either in a *synchronous* or in an *asynchronous* fashion—the agents choose.

Summing up, TuCSoN coordination paradigm preserves agent autonomy by decoupling the suspensive semantics of coordination operations from their invocation semantics, thanks to the ACC abstraction. In this way, synchronous calls are always consequences of the agent own deliberation process.

CArtAgO

CArtAgO [21] is a Java-based framework and infrastructure based on the A&A (agents & artefacts) meta-model [15]. A&A exploits *artefacts* as the tools that agents use to achieve their own goals—as humans do with their tools [9]. Artefacts can be used to uniformly represent any kind of environmental resource within a MAS—sensors, actuators, databases, etc.

Even though CArtAgO does not focus on coordination, its general-purpose artefacts programming model allows coordination artefacts to be designed. Thus, a model of coordination can be devised, in particular, based on the *agent body* abstraction. By exposing an *effectors* API and a *perception* API, CArtAgO agent bodies are the architectural components enabling (and decoupling) agent interactions with artefacts. By exploiting the effectors API, current agent activity is *suspended* until an event reporting the action *completion* is received: then, the corresponding activity resumed. Even if one activity is suspended, the agent *is not*: its working cycle can continue processing percepts and executing other actions related to other activities.

Mediation by agent bodies is the mechanism preserving agent autonomy in CArtAgO by uncoupling action suspension from caller agent suspension.

3 Autonomy-Preserving Integration Approaches

This section tackles the issue of preserving agent autonomy when integrating objective and subjective coordination at both the conceptual level – according to the models of autonomy and coordination – and the technological level [16, 20].

In [20], CArtAgO is integrated with three different agent development frameworks: Jason [3], 2APL [6] and simpA [22]. There, CArtAgO is proposed as a framework to enable and promote artefact-based interaction of heterogeneous agents. Nevertheless, authors *de facto* integrate subjective and objective coordination: in fact, by allowing Jason, 2APL, and simpA agents to exploit CArtAgO artefacts, they make it possible to build & use coordination artefacts, effectively integrating the message-based (subjective) coordination capabilities of agents with the artefact-based (objective) ones. The approach taken in [20] is an example of *autonomy-preserving integration*: e.g., in the case of Jason-CArtAgO, Jason intentions suspension mechanism is successfully integrated with CArtAgO artefacts by exploiting CArtAgO agent body abstraction. In particular, whenever a Jason agent requests execution of an operation on a CArtAgO artefact, the caller intention is automatically suspended until the "effector feedback" is received. Thus, nothing can hinder Jason agent autonomy if they *simultaneously* operate on artefacts while exchanging messages with other agents.

In [16], integration between JADE and TuCSoN technologies is successfully achieved, allowing JADE agents to exploit TuCSoN coordination services as part of the JADE platform—however, without preserving autonomy. JADE model of autonomy and TuCSoN model of coordination were not considered: in fact, if a coordination operation gets suspended, the caller behaviour is unavoidably suspended, too, because of its single thread of control being stuck waiting for operation completion. This inevitably leads to the suspension of all other behaviours the agent is (possibly) concurrently executing. Roughly speaking, the agent choice to rely on objective coordination may affect its ongoing subjective coordination activities. This is a clear example of an artificial dependency (unintentionally) created by a "non autonomy-preserving" approach—as [16] is.

In the remainder of this section, an autonomy-preserving integration called TuCSoN4JADE is presented, which successfully solves such an issue.

3.1 Preserving Autonomy in TuCSoN4JADE

The first step to integrate TuCSoN and JADE is to implement TuCSoN as a JADE *service*, actually following the work in [16]. The main novelty here concerns the `BridgeToTucson` class, as the component mediating all the interactions between JADE and TuCSoN. In particular, it offers two methods for invoking coordination operations, one for each *invocation semantics* JADE agents may choose:

`synchronousInvocation()` | lets agents invoke TuCSoN coordination operations *synchronously w.r.t. the caller behaviour*. This means the caller behaviour *only* is (possibly) suspended – and automatically resumed – as soon as the requested operation completes, not the agent as a whole—as in [16].

`asynchronousInvocation()` | lets clients *asynchronously* invoke TuCSoN coordination operations. Regardless of whether the coordination operation suspends, the agent does not, thus the caller behaviour continues.

Fig. 1 shows what happens when a synchronous operation is invoked — asynchronous invocation is not so interesting for the purpose of the paper. The "alt"-labelled frame enclosing "JADE Behaviour" entity represents the equivalent of JADE "`block()`-then-resume" programming pattern in TuCSoN4JADE. In particular, once the synchronous invocation is requested (message 2), two scenarios may occur:

completion ready | the TuCSoN operation completion event has already been generated by the TuCSoN middleware, and is already available for inspection within TuCSoN4JADE bridge (messages 3.a-4.a)

operation pending | the completion event has not reached the `BridgeToTucson` object yet—thus, from TuCSoN4JADE standpoint, the invoked operation is still pending (messages 3.b-9)

In the second case, the behaviour blocks (step 3b), waiting to be automatically resumed by TuCSoN4JADE as soon as the operation completion becomes available. Meanwhile, `BridgeToTucson` delegates execution of the operation to its associated ACC. Once such operation is finally completed, steps 7-9 cause the caller behaviour to resume. Back to the first case, we understand how TuCSoN4JADE autonomy-preserving integration technically works. We know when JADE behaviour is re-scheduled, its `action()` method re-starts *from the beginning*, thus, method `synchronousInvocation()` is re-invoked. The whole TuCSoN4JADE machinery works because such method internally (thus transparently) checks if the completion of the operation just invoked is already available: only if it is not, the whole path 3.b-9 is executed. In case it is available, `BridgeToTucson` immediately sends completion event back to the caller behaviour (step 3a).

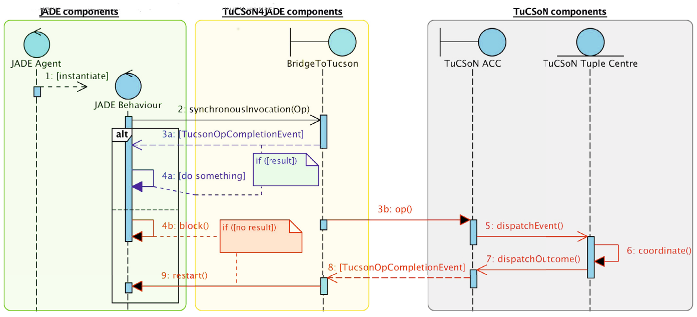

Fig. 1 TuCSoN4JADE *autonomy-preserving integration*, allowing JADE model of autonomy and TuCSoN model of coordination to integrate

3.2 Showcasing TuCSoN4JADE*: The "Book Trading" Example*

The "book trading" example is included in JADE distribution to showcase support to FIPA interaction protocols—thus, subjective coordination. In short, n seller agents advertise their catalogue of books, whereas m buyer agents browse such catalogues looking for books. The whole interaction takes the form of the well-known ContractNet protocol: buyers start a call-for-proposals, sellers reply with actual proposals, buyers choose which one to accept, the purchase is carried out. A fundamental requirement is that sellers should stay reactive to call-for-proposals even in the middle of a purchase transaction—otherwise they could lose potential revenues. We call *concurrency property* such a requirement. In the following, we take the book trading example as a paradigmatic example showcasing the practical relevance of autonomy-preserving integration approaches.

In particular, we re-think the ContractNet protocol by integrating objective and subjective coordination approaches: tuple-based call-for-proposals with message-based purchase. In fact, since the call-for-proposals should reach all the sellers, it is more efficient to put a single "call-for-proposals tuple" in a shared "contract-net space", rather than messaging each seller individually. On the contrary, since the purchase is typically a 1-to-1 interaction, messaging can efficiently do the job. This is not only conceptually correct, but also is more efficient – less messages, less network operations, etc. – in integrating an objective approach to coordination with a subjective one. We do so first exploiting the integration of TuCSoN and JADE proposed in [16] (Fig. 2), then using TuCSoN4JADE[2] (Fig. 3): in the former case, the concurrency property – thus, agent autonomy – is lost, whereas in the latter it is preserved as expected.

Fig. 2 depicts one possible instance of the run-time interactions between a given seller and a given buyer. In particular, the seller is replying to a previous call-for-

[2] The code is available as part of the TuCSoN4JADE distribution, downloadable from `http://bitbucket.org/smariani/tucson/downloads`

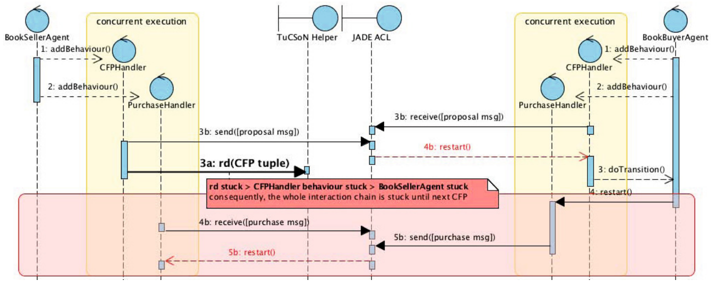

Fig. 2 Non autonomy-preserving approach taken in [16]: `rd` suspensive semantics extends to the caller behaviour, then to the caller agent, blocking all its activities

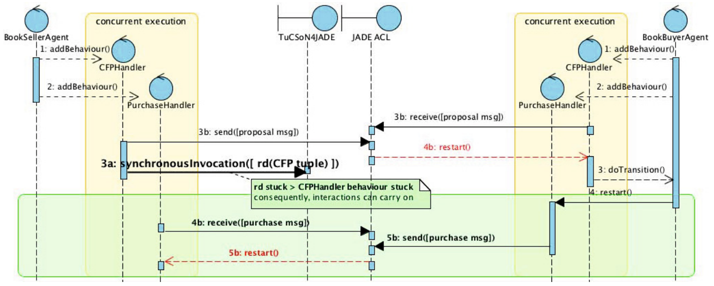

Fig. 3 TuCSoN4JADE autonomy-preserving approach: `rd` suspensive semantics is confined to the caller behaviour only, then the caller agent can carry on its other activities

proposals (message 3b). Meanwhile, it is also ready to serve new incoming call-for-proposals (3a). Here is the problem: the suspensive coordination operation `rd` gets stuck until a call-for-proposals is issued by a buyer. This is fine: it is exactly for this suspensive semantics that the LINDA model works. What is not so fine is the non autonomy-preserving approach taken by the `TucsonHelper` class in [16]: the `rd` is stuck on a network-level call and no "defensive" programming mechanism has been implemented to shield the caller behaviour. Thus it is stuck too, hindering the caller agent from scheduling other behaviours in the meanwhile—in particular, the "purchase" interaction chain (4b-5b) cannot carry on until a new call-for-proposals is issued.

Fig. 3 depicts the same scenario programmed upon the TuCSoN4JADE bridge, preserving autonomy. Since the `rd` call is shielded by a proper mechanism within the bridge, the suspensive semantics is confined to the caller behaviour. This means that only the caller behaviour is suspended – using the proper mechanisms provided by JADE, e.g. method `block()` – whereas other activities can carry on concurrently—e.g., the purchase transaction already in place (4b-5b).

The general applicability of the ContractNet protocol and its suitability for implementation as an "hybrid" protocol, drawing from both objective and subjective approaches, makes a correct integration of the two even more relevant in the context of agent development frameworks and coordination technologies.

4 Conclusion

Our goal is not just to show how JADE and TuCSoN were better integrated w.r.t. [16]—technically, it may even be seen as just a smarter implementation of the well-known OO "bridge pattern". Instead, we aim at stressing how technology-level details may have deep consequences on the higher levels of abstraction, whenever the models (possibly implicitly) brought about by technologies are not properly accounted for and understood. In particular, we demonstrate how the models of autonomy and coordination promoted by agent development frameworks may hamper an essential feature of agents: autonomy. Even though we discussed just a few agent-oriented frameworks, the issue of autonomy-preserving approaches in integrating subjective and objective coordination is quite a general one—thus, further work will be devoted to analyse other frameworks.

References

1. Bellifemine, F., Caire, G., Trucco, T., Rimassa, G.: Jade programmer's guide. Jade version 3 (2002)
2. Bellifemine, F.L., Poggi, A., Rimassa, G.: JADE–a FIPA-compliant agent framework. In: 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 1999), April 19-21, pp. 97–108. The Practical Application Company Ltd., London (1999)
3. Bordini, R.H., Hübner, J.F., Wooldridge, M.J.: Programming Multi-Agent Systems in AgentSpeak using Jason. John Wiley & Sons (October 2007)
4. Castelfranchi, C., Cesta, A., Conte, R., Miceli, M.: Foundations for interaction: The dependence theory. In: Torasso, P. (ed.) AI*IA 1993. LNCS, vol. 728, pp. 59–64. Springer, Heidelberg (1993)
5. Ciancarini, P., Omicini, A., Zambonelli, F.: Multiagent system engineering: The coordination viewpoint. In: Jennings, N.R. (ed.) ATAL 1999. LNCS (LNAI), vol. 1757, pp. 250–259. Springer, Heidelberg (2000)
6. Dastani, M., Meyer, J.-J.C.: A practical agent programming language. In: Dastani, M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS (LNAI), vol. 4908, pp. 107–123. Springer, Heidelberg (2008)
7. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems 7(1), 80–112 (1985)
8. Hexmoor, H., Castelfranchi, C., Falcone, R. (eds.): Agent Autonomy, Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 7. Springer (2003)
9. Nardi, B.: Context and Consciousness: Activity Theory and Human-computer Interaction. MIT Press (1996)
10. O'Hare, G.M., Jennings, N.R. (eds.): Foundations of Distributed Artificial Intelligence. Sixth-Generation Computer Technology. John Wiley & Sons (April 1996)

11. Omicini, A.: Towards a notion of agent coordination context. In: Marinescu, D.C., Lee, C. (eds.) Process Coordination and Ubiquitous Computing, ch. 12, pp. 187–200. CRC Press, Boca Raton (2002)
12. Omicini, A., Denti, E.: From tuple spaces to tuple centres. Science of Computer Programming 41(3), 277–294 (2001)
13. Omicini, A., Ossowski, S.: Objective versus subjective coordination in the engineering of agent systems. In: Klusch, M., Bergamaschi, S., Edwards, P., Petta, P. (eds.) Intelligent Information Agents. LNCS (LNAI), vol. 2586, pp. 179–202. Springer, Heidelberg (2003)
14. Omicini, A., Ricci, A., Viroli, M.: Coordination artifacts as first-class abstractions for MAS engineering: State of the research. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds.) SELMAS 2005. LNCS, vol. 3914, pp. 71–90. Springer, Heidelberg (2006)
15. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems 17(3), 432–456 (2008)
16. Omicini, A., Ricci, A., Viroli, M., Cioffi, M., Rimassa, G.: Multi-agent infrastructures for objective and subjective coordination. Applied Artificial Intelligence 18(9-10), 815–831 (2004)
17. Omicini, A., Zambonelli, F.: Coordination for Internet application development. Autonomous Agents and Multi-Agent Systems 2(3), 251–269 (1999)
18. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
19. Ricci, A., Omicini, A., Denti, E.: Activity Theory as a framework for MAS coordination. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2002. LNCS (LNAI), vol. 2577, pp. 96–110. Springer, Heidelberg (2003)
20. Ricci, A., Piunti, M., Acay, L.D., Bordini, R.H., Hübner, J., Dastani, M.: Integrating artifact-based environments with heterogeneous agent-programming platforms. In: 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), May 12-16, pp. 225–232. IFAAMAS, Estoril (2008)
21. Ricci, A., Viroli, M., Omicini, A.: CArtAgO: A framework for prototyping artifact-based environments in MAS. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2006. LNCS (LNAI), vol. 4389, pp. 67–86. Springer, Heidelberg (2007)
22. Ricci, A., Viroli, M., Piancastelli, G.: simpA: A simple agent-oriented Java extension for developing concurrent applications. In: Dastani, M., El Fallah Seghrouchni, A., Leite, J., Torroni, P. (eds.) LADS 2007. LNCS (LNAI), vol. 5118, pp. 261–278. Springer, Heidelberg (2008)
23. Viroli, M., Omicini, A.: Coordination as a service. Fundamenta Informaticae 73(4), 507–534 (2006)