Blending Event-Based and Multi-Agent Systems Around Coordination Abstractions

Andrea Omicini[1(✉)], Giancarlo Fortino[2], and Stefano Mariani[1]

[1] Alma Mater Studiorum–Università di Bologna, Italy
{andrea.omicini, s.mariani}@unibo.it
[2] Università della Calabria, Rende (CS), Italy
g.fortino@unical.it

Abstract. While event-based architectural style has become prevalent for large-scale distributed applications, multi-agent systems seemingly provide the most viable abstractions to deal with complex distributed systems. In this position paper we discuss the role of coordination abstractions as a basic brick for a unifying conceptual framework for agent-based and event-based systems, which could work as the foundation of a principled discipline for the engineering of complex software systems.

Keywords: Multi-agent systems · Event-based systems · Coordination models · TuCSoN

1 Introduction

In order to address some of the most common sources of *accidental complexity* – such as distributed interaction and large-scale concurrency [2] – the *event-based* architectural style has become prevalent for large-scale distributed applications in the last years [10]. At the same time, multi-agent systems (MAS) are expected to provide the most viable abstractions to deal with the modelling and engineering of complex software systems [14,15]. As a result, MAS and event-based system (EBS) stand nowadays as the two most likely candidate paradigms for modelling and engineering complex systems—the targets of many research activities on coordination models and technologies, too.

The relevance of interaction issues in both MAS and EBS suggests that coordination abstractions and mechanisms could play an essential role in making agent-based and event-based models coexist without harming conceptual integrity of systems. Starting from the essential of both paradigms, we point out the role of coordination in a unifying conceptual framework for MAS and EBS, which could work in principle as the foundation of a coherent discipline for the modelling and engineering of complex software systems.

2 MAS as Coordinated Systems

A common way to look at MAS is to interpret them according to the main first-class abstractions: agents, societies, and environment [34].

T. Holvoet and M. Viroli (Eds.): COORDINATION 2015, LNCS 9037, pp. 186–193, 2015.
DOI: 10.1007/978-3-319-19282-6_12

Agents are computational entities whose defining feature is *autonomy* [24]. Agents model *activities* for the MAS, expressed through their *actions* along with their motivations—namely, the *goals* that determine and explain the agent's course of actions. When goals are explicitly represented through *mentalistic abstractions* – as in the case of BDI agent architectures [28] – *intelligent agents* [35] are involved, which set their course of actions according to their beliefs, desires, goals, intentions, available actions, and plans.

A critical issue in MAS is *handling dependencies* between agents: that is, understanding how (intelligent) agent actions mutually interfere when each agent aims at pursuing its own goal, and ruling them so as to make MAS achieve its overall system goal. Handling dependencies is first of all a *coordination* problem [16]. Through the notion of *social action* [4], MAS capture dependencies in terms of agent *societies*, built around *coordination artefacts* [23]. Societies represent then the ensembles where the collective behaviours of the MAS are coordinated towards the achievement of the overall system goals. Generally speaking, *coordination models* are the most suitable tools to harness complexity in MAS [6], as they are explicitly meant to provide the *coordination media* that "glue" agents together [12,5] by governing agent interaction in a MAS [33].

Besides agents and societies, *environment* is an essential abstraction for MAS modelling and engineering [34], to be suitably represented, and related to agents. The notion of environment captures the unpredictability of the MAS context, by modelling the external resources and features relevant for the MAS, along with their dynamics. 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 [29] – this leads to the requirement of *situatedness* for agents and MAS, often translated into the need of being sensitive to *environment change* [9]. This basically means dependency, again: so, agent behaviour should be affected by environment change.

In all, this means that *(i)* things happen in a MAS because of either agent activity or environment change, *(ii)* complexity arises from both social and situated interaction. Also, this suggests that coordination – in charge of *managing dependencies* [16] – could be used to deal with both forms of dependency in a uniform way; so, furthermore, that coordination artefacts could be exploited to handle both social and situated interaction [17].

3 EBS as Coordinated Systems

According to [10], an EBS is "a system in which the integrated components communicate by generating and receiving *event notifications*" where an *event* is an occurrence of a happening relevant for the system – e.g., a state change in some component –, and a *notification* is the reification of an event within the system, and provides for the event description and data. Components in EBS basically act as either *producers* or *consumers* of notifications: producers *publish* notifications, and provide an *output interface* for subscription; consumers *subscribe* to notifications as specified by producers. According to the *event-based* *architectural style*, producers and consumers do not interact directly, since their interaction is mediated by the *event bus*, which abstracts away all the complexity of the *event notification service*.

In *distributed event-based systems* (DEBS) [19], a fundamental issue is represented by *distributed notification routing*, that is, the way in which notifications are routed to distributed consumers. Issues such as *event aggregation* and *transformation* have to be addressed by making individual event notifications meaningful for consumers. Relationships between events should be detected, and event *hierarchies* could be required to provide for different levels of abstraction.

In the overall, EBS are basically *coordinated* systems, where coordination is event-based [18]: process activities are mostly driven by event notifications generated by producers; transformed, aggregated, filtered, distributed by the event bus; and finally interpreted and used by consumers. Producer / consumer coordination is then *mediated* by the event bus, working as the system *coordinator*, which encapsulates and possibly automates most of the coordination activities in an EBS. As an aside, it should be noted that role of the event bus in EBS typically raises the well-known issues of the *inversion of control*: that is, control over the logic of program execution is somehow inverted [13].

4 EBS and MAS: Towards a Unifying Framework

Following [17], three are the steps for integrating MAS and EBS: recognising the *sources of events*, defining the *boundary artefacts* mediating the interaction with the event sources, and providing expressive *event-based coordination* models.

The first step is looking at agents and environment as event sources. MAS could then be seen as EBS where agents encapsulate *internal events*, while environment models *external events* through dedicated abstractions – environment *resources* – capturing the unpredictable dynamics of relevant external events. Dually, producers in an EBS are to be classified as either agents – if responsible for the designed, internal events – or environment resources—if used to model external, unpredictable events. This induces a higher-level of expressiveness in EBS: since agents encapsulate control along with the criteria for its management – expressed in terms of high-level, mentalistic abstractions –, articulated *events histories* can be modelled along with their *motivations*. In addition, since MAS environment is modelled as a first-class event-based abstraction, all *causes of change* and disruption in a MAS are modelled in a uniform way as *event prosumers* (producers and consumers)—thus improving conceptual integrity.

The second main step deals with the need for a *general event model*, requiring architectural abstractions mediating between event producers and the whole system, aimed at uniformly handling hugely-heterogeneous event sources—both agents and resources. Denoted as *boundary artefacts*, they make it possibile to translate every sorts of occurrences into a uniform system of notifications according to a common event model. This is, for instance, how Agent Communication Contexts [8] and Agent Coordination Contexts [21] work for agents, and how *event mediators* (or, *correlators*) work in the Cambridge Event Architecture [1].

Thus, boundary artefacts could be conceived *(i)* in EBS as the abstraction mediating between components and the event bus, accounting for the many diverse models for data in event notifications, *(ii)* in MAS as the constrainers for agent interaction, accounting for environment diversity and agent autonomy [33].

5 EBS and MAS: The Role of Coordination

If agents and environment work as event prosumers, coordination abstractions should deal with interaction of any sort – agent-agent, agent-environment, environment-environment interaction – taking care of their mutual dependencies, by coordinating the many resulting flows of events [16].

According to [10], the potential of event-based coordination is recognised both in academia and industry, and there exists a considerable amount of related related literature on event notification systems. In fact, a number of *event-based middleware* providing such services (e.g., JEDI [7]), as well as a number of *event-based coordination* models [27,31], technologies [11], and formalisms [20,32], witness the role of event-based middleware in the engineering of complex distributed systems, as well as the event-based nature of the most relevant coordination models, including tuple-based ones [20].

Along this line, the third step in the integration of MAS and EBS is the comprehension that coordination media [5] can handle multiple event flows [25] according to their mutual dependencies in both MAS and EBS. From the MAS viewpoint, this means that the role of coordination models in MAS [6] is to provide event-driven coordination media governing event coordination in MAS. From the EBS viewpoint, coordination in EBS is event-based [18], and the event bus and service work as the system coordinators. This means that coordination media could work as the core for an event-based architecture, and that EBS could be grounded in principle upon a suitably-expressive coordination middleware, designing the event bus around the coordination services [30].

As a result, since all events are uniformly represented through the same general event model, coordination artefacts can be used to deal with both social and situated dependencies, governing every sorts of interaction through the same set of coordination abstractions, languages, and mechanisms [17]—thus enforcing conceptual integrity. Then, coordination artefacts provide a specific computational model for dealing with event observation, manipulation, and coordination—which should make life easier for programmers and engineers.

In the context of EBS, coordination media provide a suitable way to automatise event handling, and to encapsulate the logic for the coordination of multiple related flows of events, thus counterfeiting the negative effects of inversion of control on the large scale for EBS.

6 Case Study: TuCSoN Coordination as Event-Based

The TuCSoN coordination model and infrastructure [26] can be used to illustrate in short the role of coordination in blending MAS and EBS, in particular pointing out the notions of boundary and coordination artefacts.

In detail, the basic TuCSoN architecture can be represented as in Figure 1, and explained in terms of the following MAS-EBS components.

Agents. A TuCSoN agent is any computational entity exploiting TuCSoN coordination services. To act within a TuCSoN-coordinated MAS, agents should obtain an ACC from the TuCSoN node. Any action from any agent towards the MAS – either social or situated – is *mediated* by its associated ACC.

ACC. Agent coordination contexts [21]) represent TuCSoN boundary artefacts devoted to agents. ACC both *enable* and *constraint* agents interactions, mapping every agent operation into events asynchronously dispatched to tuple centres. ACC thus *decouple* agents from MAS in control, reference, space, and time.

Probes. TuCSoN environmental resources. They are handled as sources of perceptions (*sensors*) or makers of actions (*actuators*) in a uniform way. Probes do not directly interact with the MAS, but through *mediation* of their transducer.

Transducers. The boundary artefacts devoted to probes [3]. Each probe is assigned to a transducer, specialised to handle events from that sort of probe, and to act on probes through situation operations. Transducers thus decouple probes from tuple centres in terms of control, reference, space and time.

Events. TuCSoN adopts and generalises the ReSpecT *event model* [22]. ReSpecT is the logic-based language used to program the behaviour of TuCSoN tuple centres [22]. ACC and transducers translate external events (activities and

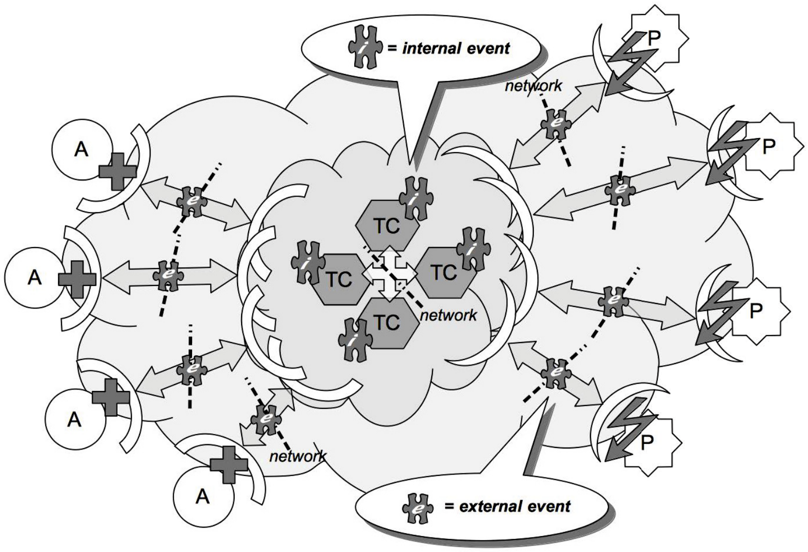

Fig. 1. TuCSoN event-based architecture

change) into internal events that tuple centres can handle to implement the policies required for MAS coordination. Thus, internal events essentially correspond to event notifications in standard EBS.

Tuple Centres. Tuple centres [22] constitute TuCSoN architectural component implementing coordination artefacts, thus in charge of managing dependencies. As such, they are meant to govern both social and situated interactions [17]. By adopting ReSpecT tuple centres, TuCSoN relies on *(i)* the ReSpecT language to program coordination laws, and *(ii)* the ReSpecT situated event model to implement events [3].

By looking at a TuCSoN-coordinated MAS with a event-based perspective,

– ACC and transducers are the boundary artefacts representing agents and environment, respectively, in the MAS, by translating activities and changes in a common event (notification) model;
– tuple centres are the coordination artefacts dealing with both social and situated dependencies by making it possible to program the coordination of events of any sorts in a clean and uniform way.

Under such a perspective, TuCSoN already provides in some way both a model and a technology to engineer coordinated MAS as EBS. Essentially, this means that when using TuCSoN for the coordination of a distributed system, either perspectives – event-based and agent-based – can be adopted by engineers according to their specific design needs, and blended together in a coherent way around the coordination abstractions provided by the TuCSoN model and middleware.[1]

7 Conclusion

Many large-scale distributed systems are nowadays designed and developed around event-based methods and technologies. At the same time, agent-based abstractions (and, in spite of their limited maturity, agent technologies, too) are more and more adopted to face the intricacies of complex systems engineering, in particular when requirements such pervasiveness, intelligence, mobility, and the like, have to be addressed. Altogether, this suggests that a conceptual framework blending together abstractions and technologies from both EBS and MAS could represent a fundamental goal for the research on complex system engineering.

In this position paper we suggest that a fundamental role in such a conceptual framework could be played by coordination models and technologies, with the focus on coordination artefacts working as both event-based and agent-based abstractions. Coordination models and middleware could then provide the technical grounding for a principled, comprehensive methodology for complex system engineering, allowing for the integration of event-based and agent-based tools and techniques without harming conceptual integrity.

[1] http://tucson.unibo.it

References

1. Bacon, J., Moody, K., Bates, J., Heyton, R., Ma, C., McNeil, A., Seidel, O., Spiteri, M.: Generic support for distributed applications. Computer 33(3), 68–76 (2000)
2. Brooks, F.P.: No Silver Bullet Essence and Accidents of Software Engineering. Computer 20(4), 10–19 (1987)
3. Casadei, M., Omicini, A.: Situated tuple centres in ReSpecT. In: Shin, S.Y., Ossowski, S., Menezes, R., Viroli, M. (eds.) 24th Annual ACM Symposium on Applied Computing (SAC 2009), vol. III, pp. 1361–1368. ACM, Honolulu (2009)
4. Castelfranchi, C.: Modelling social action for AI agents. Artificial Intelligence 103(1-2), 157–182 (1998)
5. Ciancarini, P.: Coordination models and languages as software integrators. ACM Computing Surveys 28(2), 300–302 (1996)
6. Ciancarini, P., Omicini, A., Zambonelli, F.: Multiagent system engineering: The coordination viewpoint. In: Jennings, N.R., Lespérance, Y. (eds.) Intelligent Agents VI. LNCS (LNAI), vol. 1757, pp. 250–259. Springer, Heidelberg (2000)
7. Cugola, G., Di Nitto, E., Fuggetta, A.: The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. IEEE Transactions on Software Engineering 27(9), 827–850 (2001)
8. Di Stefano, A., Pappalardo, G., Santoro, C., Tramontana, E.: The transparent implementation of agent communication contexts. Concurrency and Computation: Practice and Experience 18(4), 387–407 (2006)
9. Ferber, J., Müller, J.P.: Influences and reaction: A model of situated multiagent systems. In: Tokoro, M. (ed.) 2nd International Conference on Multi-Agent Systems (ICMAS 1996), pp. 72–79. AAAI Press, Tokio (1996)
10. Fiege, L., Mühl, G., Gärtner, F.C.: Modular event-based systems. The Knowledge Engineering Review 17(4), 359–388 (2002)
11. Freeman, E., Hupfer, S., Arnold, K.: JavaSpaces Principles, Patterns, and Practice: Principles, Patterns and Practices. The Jini Technology Series. Addison-Wesley Longman (June 1999)
12. Gelernter, D., Carriero, N.: Coordination languages and their significance. Communications of the ACM 35(2), 97–107 (1992)
13. Haller, P., Odersky, M.: Event-based programming without inversion of control. In: Lightfoot, D.E., Ren, X.-M. (eds.) JMLC 2006. LNCS, vol. 4228, pp. 4–22. Springer, Heidelberg (2006)
14. Jennings, N.R.: On agent-based software engineering. Artificial Intelligence 117(2), 277–296 (2000)
15. Jennings, N.R.: An agent-based approach for building complex software systems. Communications of the ACM 44(4), 35–41 (2001)
16. Malone, T.W., Crowston, K.: The interdisciplinary study of coordination. ACM Computing Surveys 26(1), 87–119 (1994)
17. Mariani, S., Omicini, A.: Coordinating activities and change: An event-driven architecture for situated MAS. Engineering Applications of Artificial Intelligence 41, 298–309 (2015)
18. Milicevic, A., Jackson, D., Gligoric, M., Marinov, D.: Model-based, event-driven programming paradigm for interactive Web applications. In: 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward! 2013), pp. 17–36. ACM Press, New York (2013)

19. Mühl, G., Fiege, L., Pietzuch, P.: Distributed Event-Based Systems. Springer, Heidelberg (2006)
20. Omicini, A.: On the semantics of tuple-based coordination models. In: 1999 ACM Symposium on Applied Computing (SAC 1999), pp. 175–182. ACM, New York (1999)
21. Omicini, A.: Towards a notion of agent coordination context. In: Marinescu, D.C., Lee, C. (eds.) Process Coordination and Ubiquitous Computing, chap. 12, pp. 187–200. CRC Press, Boca Raton (2002)
22. Omicini, A., Denti, E.: From tuple spaces to tuple centres. Science of Computer Programming 41(3), 277–294 (2001)
23. 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)
24. 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)
25. Omicini, A., Ricci, A., Zaghini, N.: Distributed workflow upon linkable coordination artifacts. In: Ciancarini, P., Wiklicky, H. (eds.) COORDINATION 2006. LNCS, vol. 4038, pp. 228–246. Springer, Heidelberg (2006)
26. Omicini, A., Zambonelli, F.: Coordination for Internet application development. Autonomous Agents and Multi-Agent Systems 2(3), 251–269 (1999)
27. Papadopoulos, G.A., Arbab, F.: Coordination models and languages. In: Zelkowitz, M.V. (ed.) The Engineering of Large Systems. Advances in Computers, vol. 46, pp. 329–400. Academic Press (1998)
28. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. In: Lesser, V.R., Gasser, L. (eds.) 1st International Conference on Multi Agent Systems (ICMAS 1995), pp. 312–319. The MIT Press, San Francisco (1995)
29. Suchman, L.A.: Situated actions. In: Plans and Situated Actions: The Problem of Human-Machine Communication, chap. 4, pp. 49–67. Cambridge University Press, New York (1987)
30. Viroli, M., Omicini, A.: Coordination as a service. Fundamenta Informaticae 73(4), 507–534 (2006); special issue: Best papers of FOCLASA 2002
31. Viroli, M., Omicini, A., Ricci, A.: On the expressiveness of event-based coordination media. In: Arabnia, H.R. (ed.) International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2002), vol. III, pp. 1414–1420. CSREA Press, Las Vegas (2002)
32. Viroli, M., Ricci, A.: Tuple-based coordination models in event-based scenarios. In: 22nd International Conference on Distributed Computing Systems, Workshop Proceedings, pp. 595–601. IEEE CS (2002)
33. Wegner, P.: Coordination as constrained interaction. In: Hankin, C., Ciancarini, P. (eds.) COORDINATION 1996. LNCS, vol. 1061, pp. 28–33. Springer, Heidelberg (1996)
34. Weyns, D., Omicini, A., Odell, J.J.: Environment as a first-class abstraction in multi-agent systems. Autonomous Agents and Multi-Agent Systems 14(1), 5–30 (2007)
35. Wooldridge, M.J., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review 10(2), 115–152 (1995)