Coordination of Software Agents: Models and Languages

Giacomo Cabri[1(✉)], Letizia Leonardi[2], Stefano Mariani[3], and Franco Zambonelli[3]

[1] Dipartimento FIM, Università di Modena e Reggio Emilia, Via Campi 213/B, 41125 Modena, Italy
giacomo.cabri@unimore.it
[2] Dipartimento DIEF, Università di Modena e Reggio Emilia, Via Vivarelli 10, 41125 Modena, Italy
letizia.leonardi@unimore.it
[3] Dipartimento DISMI, Università di Modena e Reggio Emilia, Via G. Amendola 2, 42122 Reggio Emilia, Italy
{stefano.mariani,franco.zambonelli}@unimore.it

Abstract. Software agents are autonomous components that perform tasks on the behalf of users. In many contexts, agents do not exist in isolation, but they must interact with other agents, leading to multiagent systems (MASs). No matter whether they collaborate or compete in MASs, coordination is needed to rule their interactions and to support them in their activities. In this chapter, we present the agent coordination models defined in the last 25 years taking the contributions to the WOA workshops as fil rouge, along with the languages that have been proposed to allow agent coordination. In addition, we sketch how the research on agent coordination has paved the way to be applied in several application fields, such as autonomic computing, self-organizing systems, energy management, and autonomous vehicles.

Keywords: Coordination · multi-agent system · models · languages

1 Introduction

Software agents represent a useful paradigm in the development of complex distributed systems [28]. They exhibit some specific features that make them useful for different kinds of application areas. Let us start from the feature that could perhaps be considered the most important one, i.e. *autonomy*, which enables software agents to carry out tasks on behalf of humans with the ability to take autonomous decisions. In addition to this, there are other features that turn out to be relevant to achieving the benefits of exploiting agents. In fact, software agents are *reactive*, which means that they sense and react to environment changes; moreover, they are *proactive*, which means that they have plans to achieve goals, and last but not least, they are *social*, which means that they

V. Mascardi and A. Omicini (Eds.): *The Agents Journey*, LNCS 16395, pp. 127–146, 2026.
https://doi.org/10.1007/978-3-032-22940-3_5

interact with other agents and the environment in which they act. The *sociality* feature of the agent enables building systems composed of several agents, which allows the definition of multi-agent systems (MASs). Therefore, MASs are the natural way to design complex distributed systems that determine two different scenarios. On the one hand, a MAS can be considered from an individualistic perspective: an aggregation of agents that simply interact with each other; in this scenario, the resulting MAS is closed, meaning that participant agents must be designed only referring to the specific problem. On the other hand, a MAS can be used to determine an open scenario to provide a conceptual model for vast organizations. In the latter scenario, the agent sociality not only enables technical powers but also enables to model, in the agents, a behavior very similar to the human one. This calls for ruling out the interactions among agents in some way. In particular, agents can compete to exploit shared resources and/or can exploit each other features to achieve self-interested goals (competitive agents); nevertheless, agents can collaborate to carry out a common goal (cooperative agents); in both cases, a coordination model/language is needed. Another interesting feature of agents can be the *mobility*, which allows agents to move to different execution contexts and introduces some interesting challenges in agent coordination.

The aim of this chapter is to provide the reader with a wide illustration of the approaches present in the literature with regard to agent coordination. This survey poses a particular emphasis to the approaches that have been presented at the Workshop on Objects and Agents (WOA) over the years to underline that the Italian research is absolutely in line with the research carried out at the international level. Our analysis on agent coordination ranges from the models to the languages, to finish discussing what application fields can benefit from using MASs.

More in detail, this chapter is organized as follows. In Sect. 2, we describe the *methodology* used to gather the WOA papers, summarizing them in a table. In Sect. 3, we provide a survey of the *coordination models* that have been defined in the last 25 years to model the agents' interactions; in this Section we first consider the two possible couplings of coordination, spatially and temporally, to arrive at defining another aspect that makes the coordination more active or reactive, i.e. the adaptability. In particular, we present MARS and other models that apply this point of view. In Sect. 4, we consider some specific coordination approaches that have again been defined in the last 25 years to rule the agents' interactions, but focusing in particular on the relative *coordination languages*. In Sect. 5, we include a discussion of application areas where the use of a MAS together with a specific model/language for agent coordination can be particularly relevant. Section 6 concludes the chapter.

2 Methodology

To select the WOA papers to analyze in this chapter, we started from two sources of information tracked year after year by the WOA organization: a bibliography

file tracking citation metadata about all papers presented at WOA, and a spreadsheet with abstracts of such papers. We automatically have filtered relevant contributions for the current chapter by searching for keywords "coordination model" and "coordination language", and then removing duplicates (since in some WOA editions, demo papers had the same title as main event papers). Afterwards, we manually further filtered the partial results by reading the abstracts and, when needed, the paper itself to evaluate whether the contribution is relevant to the chapter. The end result of this process is the collection of papers cited in this chapter and summarized in Table 1.

Table 1. Summary of coordination related contributions discussed at WOA. In the "Category" column, M = model, L = language, W = middleware, A = application.

Year	Refs	Brief description	Category
'98–14	[39,41,52,53]	TuCSoN, programmable tuple-based coordination	M W
2000	[11]	MARS, programmable coordination model	M L W
'02–13	[38,43,59]	ReSpecT, programmable tuples	M L
2004	[32]	TOTA, distributed, self-organising tuples	M W
2006	[45]	Coordination for autonomic computing	A
'08–12	[46,57,62,64]	SAPERE, distributed, self-organising semantic tuples	M L W
2010	[44]	Coordination for grid computing	A
'10–13	[36,47]	Bio-chemical inspired, self-organising tuples	M
2012	[14]	Coordination for energy regulation	A
'12–17	[2,4,6]	Commitment-based architecture	M W
2015	[20]	Coordination for unmanned vehicles	A
2018	[54]	Coordination for smart factories	A

3 Coordination Models

During its life, an agent is in need to coordinate its activities with other entities, let them be other agents or resources on hosting execution environments. More in particular:

- an application may be composed of several agents that cooperatively perform a task and, then, are in need of coordinating their activities;
- an agent, in case it is mobile, is usually in need to roam across remote sites to access resources, services, and even other agents there allocated.

As a case study application, let us consider a simple WWW information retrieval application. An agent is sent to a remote site to analyze the WWW pages and returns to the starting site with the URLs of the pages that contain a specific

keyword. The agent clones itself for any remote link found in the pages of interest and sends the clones to the found sites, to recursively continue the searching work. Inter-agent coordination is needed to avoid multiple visits of different agents on the same site (since it is common to find cross-references in HTML pages). The coordination of the agents with regard to the hosting execution environment aims to define a precise protocol to access and retrieve information on a site.

Although coordination models have been extensively studied in the past, the openness of the agent environment introduces new problems and needs. In this section, we define a simple taxonomy of the possible coordination models for agent applications [12]. Two main characteristics distinguish different coordination models: spatial and temporal coupling. In particular:

- *spatially* coupled coordination models require the involved entities to share a common name space; conversely, spatially uncoupled models enforce anonymous interactions;
- *temporally* coupled coordination models imply synchronization of the involved entities; conversely, temporally uncoupled coordination models achieve asynchronous interactions.

Therefore, four categories of coordination models can be derived (see Fig. 1): (i) direct, both spatially and temporally coupled; (ii) meeting-oriented, spatially uncoupled and temporally coupled; (iii) blackboard-based, spatially coupled and temporally uncoupled; (iv) Linda-like, both spatially and temporally uncoupled. In the following (Sects. 3.1 to 3.4), we detail these four coordination models.

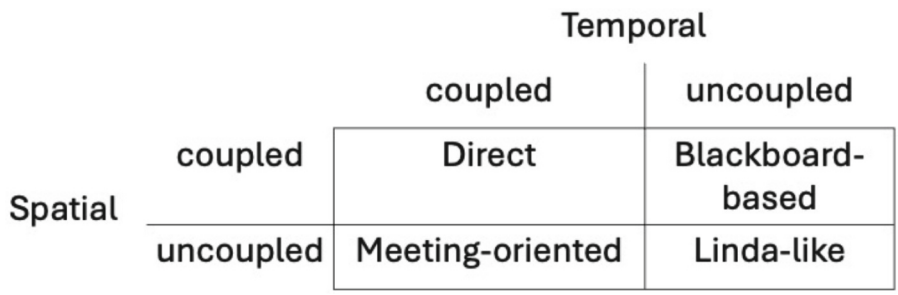

Fig. 1. Coordination Models for Agents

However, from our research work we can affirm that all these models exhibit a possible limitation. In fact, they are "passive" and therefore do not provide active support for applications. So, we decided to explore "active" or "reactive" coordination models that can introduce another aspect of coordination, i.e. *adaptability* (see Sect. 3.5).

3.1 Direct Coordination

In direct coordination models, agents can initiate a communication by explicitly naming the involved partners (spatial coupling). This usually implies their synchronization (temporal coupling). In the case of inter-agent coordination, two agents must agree on a communication protocol, typically a peer-to-peer one. In the case of access to the resources of the hosting environment, coordination usually occurs in a client-server way.

The general adoption of direct coordination models in agent applications is not suitable. Repeated interactions require stable network connections, making communication highly dependent on network reliability. In addition, wide-area communications between entities require complex and highly informed routing protocols. Finally, because several agent applications are intrinsically dynamic (through dynamic agent creation), it may be difficult to adopt a spatially coupled model in which the communication partners must be identified. In the case study application, the agents cannot know how many other agents make up the application because the agents are dynamically created depending on the found links. In addition, to establish a communication session, agents must be forced to synchronize their activities, which instead are intrinsically asynchronous and autonomous. In the presence of many agents, direct coordination models can be effectively exploited only for accessing local resources: a local server has to be provided as a manager, and an agent interacts with it in a client-server way. In our case study application, local WWW servers may supply HTML pages to agents.

3.2 Meeting-Oriented Coordination

In meeting-oriented coordination, agents can interact without the need to explicitly name the partners involved. Interactions occur in the context of known meeting points that agents join, either explicitly or implicitly, to communicate and synchronize with each other. Apart from always-opened meetings - which can abstract the role of servers in an execution environment – an active entity must assume the role of initiator to open a meeting point. Often, meetings are locally constrained: to avoid the problems related to non-local communication (i.e., unpredictable delay and unreliability), a meeting takes place at a given execution environment, and only local agents can participate in it. Clearly, because agents must share common knowledge of either the meeting names or of the events that force them to join a meeting, full spatial uncoupling is not achieved.

Although the meeting model partially solves the problem of exactly identifying the involved partners, it has the drawback of enforcing strict synchronization between agents. Because in many applications, the schedule and the position of agents cannot be predicted, the risk of missing interactions is very high. In the case study application, to avoid multiple visits to the same site with a meeting-oriented model, an additional agent must be introduced for each site visited. When an agent has explored a site, it creates a "meeting" agent before returning to the user site. The meeting agent is forced to reside on the creation site: as any

other agent arrives at this site, it tries to enter a meeting with the meeting agent to detect whether the site has been already visited or not. In our opinion, this solution is not generally suitable for the case study application because there is no possibility that the meeting agent will know when to die. In addition, to let an agent remain on one visited site is not safe: a malicious agent can exploit the time it is on a site to furnish private information to the external.

3.3 Blackboard-Based Coordination

In blackboard-based coordination, interactions occur through shared data spaces, local to each hosting environment, used by agents as common repositories to store and retrieve messages. Insofar as agents must agree on a common message identifier to communicate and exchange data via a blackboard, they are not spatially uncoupled.

The most significant advantage of this coordination model derives from fully temporal uncoupling: messages can be left on blackboards without needing to know neither where the corresponding receivers are nor when they will read the messages. This clearly suits a scenario in which the position and the schedules of the agents can be neither monitored nor granted easily. In addition, being any inter-agent interaction forced to be performed via a blackboard, hosting environments can easily control all interactions, thus leading to an execution environment model more secure than that of the coordination approaches described above. Our case study application can effectively exploit the blackboard model for inter-agent coordination. When an agent arrives at a site, it first checks on the local blackboard for a "marker" message, coming from another agent of the same application. If the agent reads the "marker" message, it knows that the site has already been visited; otherwise, it is in charge of placing the "marker" message on the blackboard. With regard to agent-to-hosting execution environment interactions, a blackboard can be exploited to let agents retrieve the needed information without requiring the presence of a specialized resource manager and to let the local environment provide in the blackboard all the data it wants to publish while protecting private data. In the case study application, the execution environment could provide, in the form of messages, the pathnames of all its publicly accessible files. However, there is no way for an agent to retrieve the pathnames of the HTML files only, but all messages have to be retrieved and successively selected to search those corresponding to HTML files.

3.4 Linda-like Coordination

In Linda-like coordination, the accesses to a local blackboard are based on associative mechanisms [15]: Information is organized into tuples and retrieved associatively through a pattern matching mechanism. Associative blackboards (called tuple spaces) enforce full uncoupling, requiring neither temporal agreement nor mutual knowledge to let agents coordinate.

Linda-like coordination suits agent applications well. In a wide and dynamic environment, such as the Internet, having complete and updated knowledge of

hosting execution environments and other application agents may be difficult or even impossible. Then, because agents would somehow require pattern matching mechanisms to adaptively deal with uncertainty, dynamicity, and heterogeneity, it is worthwhile to integrate these mechanisms directly into the coordination model, to simplify agent programming, and to reduce application complexity. In the testbed application, associative mechanisms may be not necessary for inter-agent interactions, because agents know which message to retrieve from the blackboard to avoid multiple visits, as shown in the previous subsection. However, let our application be composed of several types of agent, each devoted to the search for a different keyword. In this case, an associative mechanism would be necessary: an agent must check the presence of marker messages in the tuple space with the keyword field that matches the agent's own keyword; instead, it must ignore the messages left by agents searching for different keywords. With regard to agent-to-hosting execution environment interactions, if the pathnames of all public readable files are available in the tuple space, agents can simply look for tuples corresponding to pathnames matching the "html" extension.

3.5 Adaptable Coordination

As mentioned above, we further study the possibility of making the coordination medium *active* or *reactive*, to provide *adaptable* coordination and better support to the development of agent applications. In this way, the environment can become an actor in the coordination, taking an active part in the coordination process. This allows the coordination to be adaptable to the context in which it occurs, leading to *context-aware* coordination [13]. With "context" we mean a broad range of aspects, from the environment where the coordination takes place to the specific agents that need to coordinate with each other, from external information to the history of the involved agents.

In the following, we present an example of a coordination architecture that implements the idea of adaptable coordination (MARS), and then briefly present other approaches. As the readers will see, the most suitable model for facing adaptability is the Linda-like one.

MARS (Mobile Agent Reactive Spaces). MARS [11] has been proposed by the researchers of the University of Modena and Reggio Emilia. MARS is a programmable coordination architecture for Java-based mobile agents. Nevertheless, it can be associated to different agent systems with little modifications.

MARS adopts the Linda-like model and makes many independent *tuple spaces* available to the agents. It was originally conceived for *mobile* agents, but can be used with any kind of agent. Each tuple space is associated with a node and is accessed by locally executing agents. For each space, MARS provides a *metalevel tuple space*, which contains tuples that represent actions that can be executed in reaction of agents' access to the "normal" space. In this way, each MARS tuple space can "react" to accesses performed by agents that exhibit a behavior programmed by the space administrator or by the agents themselves (see Fig. 2).

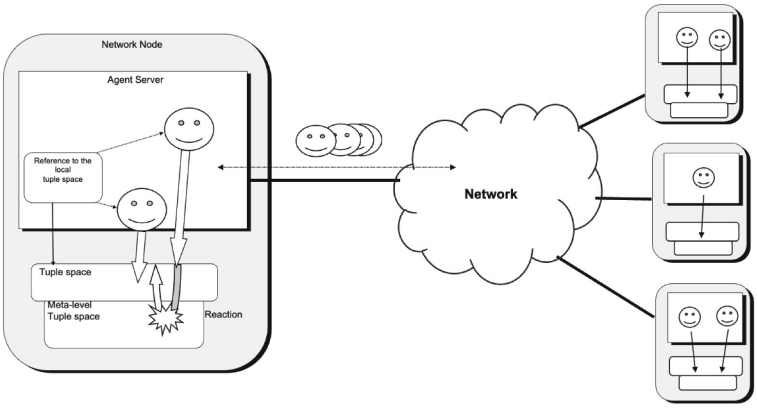

Fig. 2. The MARS coordination architecture (image adapted from [11]).

The MARS tuple space enables inter-agent coordination and allows agents to access primitive data items and references to execution environment resources. The local tuple space is the only node resource that agents can directly access, reducing the problem of dynamically binding local references to mobile entities.

Other Approaches. PageSpace [17], an architecture for interactive Web applications, uses Linda-like coordination. Both mobile and fixed agents can use its tuple spaces to store and associatively retrieve object references. In addition, agents can create tuple spaces to interact privately without affecting host execution environments. To influence the coordination activities of application agents, PageSpace is not reactive in itself but instead defines special-purpose agents to access the space and change its contents. These special-purpose agents can be limited both in monitoring interaction events and in tuning their effects.

The IBM T Spaces [61] project uses Linda-like interaction spaces as general-purpose information repositories for mobile and network applications. Rather than defining agent-oriented coordination media, the T Spaces architecture aims to provide a powerful and standard interface for accessing large amounts of information organized as a database. Therefore, the designers of T Spaces integrated a special programmability to add new behaviors to a tuple space. This occurs through new admissible operations on a tuple space rather than by programming the effects on the basic Linda operations; of course, this requires application agents to either be aware of operations available in a given tuple space or somehow dynamically acquire this knowledge.

The MOLE system [10] defines a meeting-oriented coordination model rather than a tuple-based one. MOLE meetings use shared, non-mobile objects that agents must access to send or receive messages. MOLE meetings enforce temporal uncoupling by permitting asynchronous message notification to agents and can be programmed to integrate specific policies for managing the exchanged messages. These characteristics provide an uncoupled and programmable coor-

dination model that enforces a control-oriented coordination style, in contrast to a data-oriented one, and requires the definition of meeting points at the application level.

TuCSoN [40] extends the Linda coordination model that defines *tuple centers* whose behavior can be programmed by a specific language. TuCSoN is detailed in Sect. 4.3.

4 Coordination Languages

Coordination *models* abstractly define the concepts necessary to give shape to a desired interaction paradigm, i.e., one with desired properties. For instance, the blackboard or Linda models mentioned in previous section, respectively, offer temporal, and both temporal and spatial uncoupling properties. Coordination *languages*, instead, concretely define the vocabulary and grammar of programming languages specifically meant to deal with the *interaction* space of computing, not the more common algorithmic space [60]. Of course, several of the coordination models proposed throughout the years also brought along their own coordination language to give the model an operational incarnation suitable for practical experiments, implementations, and real-world deployments.

The WOA workshop series has seen some of such languages:

- the "Tuples Over The Air" (*TOTA*) model, language, and middleware [32];
- the "Self-Aware PERvasive Ecosystems" (*SAPERE*) model, language, and middleware [46,57,62,64], originated in the context of the homonym EU FP7 project;
- the "TUple Centres Spread Over the Network" (TuCSoN) model, language, and middleware [39,41,52,53];
- the *commitment-based* languages proposed by Baldoni et al. [2,4,6];
- the "Reaction SPECification Tuples" (ReSpecT) adopted in TuCSoN and proposed by Omicini et al. [38,43,59];
- the *biochemistry-inspired* languages [36,47] that were either precursors of or inspired by the SAPERE approach already mentioned.

4.1 TOTA

TOTA is a framework that utilizes *distributed tuples* to represent contextual information and allows uncoupled interaction between the components of a distributed application. Unlike other shared data-space models, TOTA tuples are not node specific or tied to a particular data space within the network. Instead, they are put into the network and *autonomously propagate* according to the specific *propagation rules*. Consequently, TOTA tuples form a *spatially distributed data structure* that conveys both information exchange among the application components and contextual information about the distributed environment itself. In TOTA, such an environment features a peer-to-peer network of

possibly mobile nodes, each running a local instance of the TOTA middleware—as depicted in Fig. 3. The middleware maintains references to a limited set of *neighboring nodes*, and *automatically adapts* to dynamic changes caused by node mobility or failures. In scenarios like Mobile Ad Hoc Networks (MANETs), TOTA nodes identify neighbors within their wireless communication range. The propagation rules for tuples diffusion are automatically triggered by the TOTA middleware when the appropriate conditions occur (e.g., a new node enters the neighborhood, a certain tuple is added/removed to the current tuple space, and other conditions). Such rules may not only modify the location of the tuple, but also its content, allowing the creation of an *overlay* of distributed tuples meant to transmit information across the network and represent *contextual information* about the application environment in a distributed way. TOTA has been used to program coordinated motion patterns [35], implement location and content-based information access [31], implement mobile particle systems [30], in order to enable stigmergic coordination [33], and to program modular robots [34].

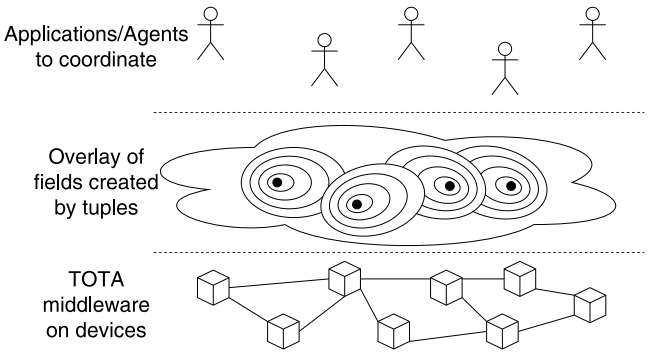

Fig. 3. The TOTA coordination model and language.

4.2 SAPERE

SAPERE [16,63] is a coordination model for multi-agent pervasive systems inspired by natural *chemical reactions*, featuring its own coordination language ("eco-laws"). The SAPERE architecture is depicted in Fig. 4. SAPERE is based on four main concepts: Live Semantic Annotations (LSAs), LSA Tuple Space, Agents, and Eco-laws. LSAs are tuples of types (name, value) that are used to store application data. LSAs belonging to a computing node are stored in a shared container named LSA Tuple Space. Each LSA is associated with an agent, such as sensors, services, or general applications, that wants to interact with the LSA space, e.g., injecting or retrieving LSAs from the LSA space. Inside the shared container, tuples react in a virtual chemical solution by using a predefined set of *coordination rules* named eco-laws, which can (i) instantiate

relationships among LSAs (Bonding eco-law), (ii) aggregate them (Aggregate eco-law), (iii) delete them (Decay eco-law), and (iv) spread them across remote LSA Tuples Spaces (Spreading eco-law). When a tuple is modified by an eco-law, its relative agent is notified. The implementation of the SAPERE middleware allowed several kinds of real distributed self-adaptive and self-organizing applications to be developed [63]. More discussion about these sort of self-organising coordination models, languages, and middleware is provided by Chapter [1].

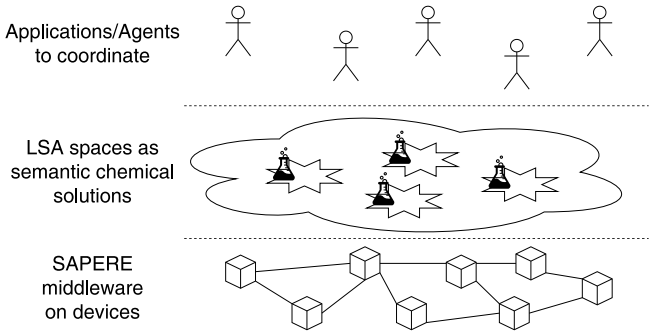

Fig. 4. The SAPERE coordination model and language.

4.3 TuCSoN

TuCSoN is a coordination model that adopts *Linda* as its core but extends it in several ways, such as by adopting nested tuples (expressed as first-order logic terms), adding primitives (i.e., bulk [55] and uniform [40]), and replacing tuple spaces with *tuple centres* [50]—that is, tuple spaces whose behavior can be programmed with a dedicated language: ReSpecT [48]. As such, the main driving concepts behind the TuCSoN model and technology are (i) *first-order logic tuples* and ReSpecT reactions to enable the declarative expression of coordination policies, (ii) asynchronous communication and coordination primitives by default (although synchronous versions are also available) to enable *full decoupling*, and (iii) programmable tuple spaces to enable full control over the *coordination policies* to be followed by the system at hand, there included security policies [19]. TuCSoN comes with a Java-based implementation that provides coordination as a service [58] in the form of a Java library (defining an API) and a middleware runtime, especially targeting distributed Java processes, but open to rational agents implemented in tuProlog [25]—see Fig. 5.

4.4 Commitments

Commitment-based communication artifacts [5] (i.e., computational tools providing functions) in multi-agent systems implement interaction protocols and monitoring functionalities—as thoroughly described in Chapter [7]. These artifacts

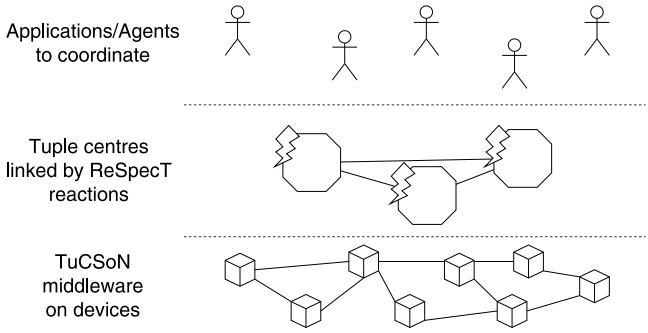

Fig. 5. The TuCSoN coordination model and language.

verify ongoing interactions for protocol adherence, detect violations, and identify violators. As *programmable* communication channels, artifacts encapsulate what commitment protocols refer to as "the social state", which captures the interaction session between parties and provides the necessary social actions for agents to enter and comply with commitments. Consequently, the artifacts encode the *coordination rules* of the protocol. A commitment, denoted as $C(x, y, r, p)$, represents a contractual relationship between a debtor (x) and a creditor (y). Specifically, the debtor commits to the realization of the consequent condition (p) when the antecedent condition (r) is satisfied. Commitments thus function as directed obligations from the debtor to the creditor. Unlike traditional obligations, commitments are subject to manipulation, for example, delegation, assignment, or release. Importantly, commitments serve a regulative purpose: social expectations dictate that agents respect the commitments that involve them, e.g. to preserve reputation. In particular, the debtor bears the responsibility for fulfilling the consequent condition. Thus, agents' behavior is influenced by the commitments present in the social state. Commitment protocols typically consist of a set of shared actions agreed upon by all interacting agents. These social actions modify the social state, for example, by adding new commitments, releasing agents from existing commitments, or satisfying commitments.

4.5 ReSpecT

The ReSpecT language [48] is rooted in first-order logic and serves to define *tuple centres*' behavior within TuCSoN. A tuple centre is a tuple space that introduces *programmability* into the coordination environment. Unlike a traditional tuple space, where the response to communication events (adding, reading, and removing tuples) is predetermined and unchangeable, a tuple centre's behavior can be changed. This flexibility is achieved by programming a collection of *specification tuples*, also known as *reactions*. ReSpecT, as a language for behavior specification, facilitates the creation of internal computations in a tuple centre, and their linkage to specific communication events. It enables the declarative pairing of events with reactions through unique logic constructs termed specification

tuples, formatted as $reaction(E, G, C)$. For an event Ev, a specification tuple $reaction(E, G, C)$ will execute a computation C upon the happening of the event Ev, provided that E and Ev "match"—in ReSpecT, the matching is defined on top of Prolog unification mechanisms [49]. Moreover, computation C is executed only if the guard predicates defined in G, which express conditions on the event or the state of the tuple centre, evaluate to `true`. ReSpecT thus allows for the procedural programming of reactions as sequences of computations (in terms of logical goals), with each computation potentially achieving success or encountering failure. Every reaction elicited by a communication event is processed prior to addressing any subsequent events. Consequently, agents experience the outcome of the communication event (e.g., invocation of a Linda primitive) and the simultaneous execution of all related responses as one unified shift in the state of the tuple center. Therefore, the impact of a communication operation on a tuple centre can be tailored to be as intricate as necessary, depending on the coordination demands of the application at hand.

4.6 Biochemical Tuple Spaces

Biochemical tuple spaces [56] are the basis of a coordination model inspired by biochemical solutions where tuples always have an associated activity or pertinency value. This value represents the concept of *chemical concentration* to determine how much a tuple can influence the coordination state. For example, a low concentration tuple would participate in the execution of coordination rules infrequently. In fact, this model incorporates coordination laws that resemble *chemical reactions*. These laws are integrated into the space of the tuples, and they cause the concentration of tuples to change over time, mirroring the behavior of chemical substances in a solution. This allows for the use of chemical patterns to bring about interesting self-organizing properties. Note that if t is a tuple specified by *in* (extract) or *rd* (read) operations, a tuple existing in the space that matches t as in Linda is looked up. However, unlike Linda, the matching function here is application dependent and returns a degree of matching (e.g. $\in [0, 1]$) instead of a yes/no: stronger matching (e.g. higher semantic matching) implies a higher *probability* of finding a certain tuple. This probabilistic aspect is the crucial mechanism that ultimately enables self-organization and adaptiveness. Furthermore, these laws include a mechanism for the diffusion of tuples, which simulates the movement of chemical substances across the boundaries of biological compartments. This adds another layer of realism to the model, making it a powerful tool for studying biochemical systems, and promotes its usage in networked systems.

Common to all these heterogeneous languages is their inspiration for other fields of research, such as physics, sociology, chemistry, made not just to appear "fancy", but to inherit some desirable properties and to attack the peculiar challenges of some application domains [29]. For example, biochemical coordination approaches directly enable and support self-organization and the resulting self-* properties [26]. Chapter [1] provides ample discussion about these kinds of self-organising coordination models, languages, and middleware.

5 Application Fields

In this section, we sketch some application fields in which the coordination of software agents can be exploited in an effective way.

Autonomic computing is a context in which agent coordination has been exploited to achieve flexible and adaptable systems [45]. In fact, the autonomic computing and multi-agent systems share different aspects that enable synergies between them. Autonomic computing requires a general purpose component model, in order to enforce autonomic behavior in both the forms of self-adaptation and self-organization, to handle "situatedness" in complex knowledge environments, and to tolerate scalable forms of dynamic aggregation. Multi-agent systems research can play a major role in the definition of such component model and, more generally, in the advance of the autonomic communication research area.

Coordination has been exploited in the field of energy regulation [14]. Since private houses can produce renewable energy, for example, by means of solar panels, owners could sell the unused energy to neighbors. A peer-to-peer model seems the most suitable for energy negotiation (i.e., between single producers and single consumers), even in a deregulated way. On the one hand, this situation could lead to advantages for both producers and consumers, but on the other hand it is very complex and dynamic. An agent-based approach can help to tackle complexity, thanks to the autonomy of agents. Their coordination can grant that the system meets defined policies, both at the producers/consumers level and at the community level. In addition, a suitable coordination enables the optimization of energy production and supply costs by means of negotiation and learning.

Another interesting field in which coordination models and languages can be exploited is the management of autonomous vehicles [37]. An autonomous vehicle can be modeled as an agent that has some goals, for example, to reach the destination as soon as possible or to park near the office. To achieve the goals, the vehicle must compete with other vehicles to exploit common resources, such as roads, intersections, and parking slots. Sometimes, they must collaborate to exploit the resources, for instance, aggregating in platoons. To complete the scenario, the local administration can have some goals, for instance reducing the pollution caused by the vehicles. In any case, their coordination is needed to regulate the movements and to apply local or global polices.

In general, self-organizing approaches take advantage of the experience in the agent world, in particular reusing the coordination models and languages in other different areas, such as computational grids [44], unmanned vehicles [20], smart factories [54].

6 Conclusion

In this chapter, we have addressed the topic of the coordination of software agents, starting from the point that the agents are *social*, and this implies their

interaction and then their coordination. We have focused on *coordination models* and *coordination languages*, about which we have reported an overview. We have reported in particular the research works presented at WOA in it 25 years. We have also sketched some application fields that can take advantage of the coordination of software agents.

After this overview work, we can provide some considerations.

First, only 23 WOA papers over almost 400 tackled the coordination issue in a sharp way. Of course, WOA addressed and addresses a broad range of topics, from agent modeling to Agent-Oriented Software Engineering, from intelligent agents to agents applications, so it is reasonable that the coordination issue concerns only a slice of the total number of papers.

The second consideration is about the fact that WOA papers on coordination models and languages are not recent. Most of them have been published before 2010. This can mean that the agent coordination is quite settled and no further specific research is needed. In any case, we believe that this requires some reflection from the WOA community.

Acknowledgments. The authors would like to thank all students and researchers who have contributed to the implementation of the models and the languages, and also the applications presented in this chapter.

Disclosure of Interests. The authors have no competing interests.

References

1. Aguzzi, G., Casadei, R., Pianini, D., Viroli, M.: Self-organisation with aggregate computing: a reflection under the lenses of multi-agent systems engineering. In: Mascardi and Omicini [42] (2026)
2. Baldoni, M., Baroglio, C.: Some thoughts about commitment protocols (position paper). In: De Paoli and Vizzari [24], pp. 68–71 (2012). http://ceur-ws.org/Vol-892/paper11.pdf
3. Baldoni, M., Baroglio, C., Bergenti, F., Garro, A. (eds.): WOA 2013 – 14th Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 1099. Turin, Italy (2013). http://ceur-ws.org/Vol-1099/
4. Baldoni, M., Baroglio, C., Capuzzimati, F.: 2COMM: a commitment-based MAS architecture. In: Baldoni et al. [3], pp. 38–57 (2013). http://ceur-ws.org/Vol-1099/paper11.pdf
5. Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Exploiting social commitments in programming agent interaction. In: Chen, Q., Torroni, P., Villata, S., Hsu, J.Y., Omicini, A. (eds.) PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, 26–30 October 2015. Proceedings. Lecture Notes in Computer Science, vol. 9387, pp. 566–574. Springer (2015). https://doi.org/10.1007/978-3-319-25524-8_39
6. Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Endowing business artifacts with a normative coordination layer. In: De Meo et al. [21], pp. 71–77 (2017). http://ceur-ws.org/Vol-1867/w13.pdf
7. Baldoni, M., Baroglio, C., Micalizio, R.: Interaction protocols: from AUML to social commitments, from artifacts to BSPL. In: Mascardi and Omicini [42] (2026)

8. Baldoni, M., Cossentino, M., De Paoli, F., Seidita, V. (eds.): WOA 2008 – 9th Workshop "From Objects to Agents". Seneca Edizioni Torino, Palermo, Italy (2008). http://www.pa.icar.cnr.it/woa08/materiali/Proceedings.pdf
9. Baldoni, M., De Paoli, F., Martelli, A., Omicini, A. (eds.): WOA 2004 – 5th Workshop "From Objects to Agents". Pitagora Editrice Bologna, Torino, Italy (2004). http://lia.deis.unibo.it/books/woa2004/atti.pdf
10. Baumann, J., Hohl, F., Rothermel, K., Straßer, M.: Mole-concepts of a mobile agent system. World Wide Web **1**, 123–137 (1998)
11. Cabri, G., Leonardi, L., Zambonelli, F.: MARS: a programmable coordination architecture for mobile agents. IEEE Internet Comput. **4**(4), 26–35 (2000). https://doi.org/10.1109/4236.865084
12. Cabri, G., Leonardi, L., Zambonelli, F.: Mobile-agent coordination models for internet applications. Computer **33**(2), 82–89 (2000). https://doi.org/10.1109/2.820044
13. Cabri, G., Leonardi, L., Zambonelli, F.: Engineering mobile agent applications via context-dependent coordination. IEEE Trans. Software Eng. **28**(11), 1039–1055 (2002). https://doi.org/10.1109/TSE.2002.1049403
14. Capodieci, N., Cabri, G., Pagani, G.A., Aiello, M.: Agent modeling of a pervasive application to enable deregulated energy markets. In: De Paoli and Vizzari [24], pp. 8–16 (2012). http://ceur-ws.org/Vol-892/paper3.pdf
15. Carriero, N., Gelernter, D.: Linda in context. Commun. ACM **32**(4), 444–458 (1989)
16. Castelli, G., Mamei, M., Rosi, A., Zambonelli, F.: Pervasive middleware goes social: the SAPERE approach. In: Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems, SASOW 2011, Ann Arbor, MI, USA, 3–7 October 2011, Workshops Proceedings, pp. 9–14. IEEE Computer Society (2011). https://doi.org/10.1109/SASOW.2011.6
17. Ciancarini, P., Tolksdorf, R., Vitali, F., Rossi, D., Knoche, A.: Coordinating multiagent applications on the WWW: a reference architecture. IEEE Trans. Software Eng. **24**(5), 362–375 (1998)
18. Cossentino, M., Sabatucci, L., Seidita, V. (eds.): WOA 2018 – 19th Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 2215. Palermo, Italy (2018). http://ceur-ws.org/Vol-2215/
19. Cremonini, M., Omicini, A., Zambonelli, F.: Coordination and access control in open distributed agent systems: the TuCSoN approach. In: Porto, A., Roman, G.C. (eds.) Coordination Languages and Models. Lecture Notes in Computer Science, vol. 1906, pp. 99–114. Springer, Cham (2000). https://doi.org/10.1007/3-540-45263-X_7. 4th International Conference (COORDINATION 2000), Limassol, Cyprus, 11–13 September 2000. Proceedings
20. De Benedetti, M., D'Urso, F., Messina, F., Pappalardo, G., Santoro, C.: Self-organising uavs for wide area fault-tolerant aerial monitoring. In: Di Napoli, C., Rossi, S., Staffa, M. (eds.) WOA 2015 – 16th Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 1382, pp. 135–141. Naples, Italy (2015). http://ceur-ws.org/Vol-1382/paper21.pdf
21. De Meo, P., Postorino, M.N., Rosaci, D., Sarnè, G.M.L. (eds.): WOA 2017 – 18th Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 1867. Scilla, RC, Italy (2017). http://ceur-ws.org/Vol-1867/
22. De Paoli, F., Di Stefano, A., Omicini, A., Santoro, C. (eds.): WOA 2006 – 7th Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 204. Catania, Italy (2006). http://ceur-ws.org/Vol-204/

23. De Paoli, F., Manzoni, S., Poggi, A. (eds.): WOA 2002 – 3rd Workshop "From Objects to Agents". Pitagora Editrice Bologna, Milano, Italy (2002). http://giuseppevizzari.github.io/WOA-proceedings-archive/woa-2002.html
24. De Paoli, F., Vizzari, G. (eds.): WOA 2012 – 13th Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 892. Milano, Italy (2012). http://ceur-ws.org/Vol-892/
25. Denti, E., Omicini, A., Ricci, A.: tuProlog: a light-weight prolog for internet applications and infrastructures. In: Ramakrishnan, I. (ed.) Practical Aspects of Declarative Languages, Lecture Notes in Computer Science, vol. 1990, pp. 184–198. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45241-9_13. 3rd International Symposium (PADL 2001), Las Vegas, NV, USA, 11–12 March 2001. Proceedings
26. Fernandez-Marquez, J.L., Serugendo, G.D.M.: From self-organizing mechanisms to design patterns to engineering self-organizing applications. In: 7th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2013, Philadelphia, PA, USA, 9–13 September 2013, pp. 267–268. IEEE Computer Society (2013). https://doi.org/10.1109/SASO.2013.21
27. Fortino, G., Garro, A., Palopoli, L., Russo, W., Spezzano, G. (eds.): WOA 2011 – 12th Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 741. Rende, Italy (2011). http://ceur-ws.org/Vol-741/
28. Jennings, N.R.: An agent-based approach for building complex software systems. Commun. ACM **44**(4), 35–41 (2001)
29. Mamei, M., Menezes, R., Tolksdorf, R., Zambonelli, F.: Case studies for self-organization in computer science. J. Syst. Archit. **52**(8–9), 443–460 (2006). https://doi.org/10.1016/J.SYSARC.2006.02.002
30. Mamei, M., Vasirani, M., Zambonelli, F.: Self-organizing spatial shapes in mobile particles: the TOTA approach. In: Brueckner, S., Serugendo, G.D.M., Karageorgos, A., Nagpal, R. (eds.) Engineering Self-Organising Systems, Methodologies and Applications [revised versions of papers presented at the Engineering Selforganising Applications (ESOA 2004) Workshop, held During the Autonomous Agents and Multi-agent Systems conference (AAMAS 2004) in New York in July 2004, and selected invited papers]. Lecture Notes in Computer Science, vol. 3464, pp. 138–153. Springer (2004). https://doi.org/10.1007/11494676_9
31. Mamei, M., Zambonelli, F.: Location-based and content-based information access in mobile peer-to-peer computing: the TOTA approach. In: Moro, G., Sartori, C., Singh, M.P. (eds.) Agents and Peer-to-Peer Computing, Second International Workshop, AP2PC 2003, Melbourne, Australia, 14 July 2003, Revised and Invited Papers. Lecture Notes in Computer Science, vol. 2872, pp. 162–173. Springer (2003). https://doi.org/10.1007/978-3-540-25840-7_17
32. Mamei, M., Zambonelli, F.: Spatial computing: the TOTA approach. In: Baldoni et al. [9], pp. 126–142 (2004). http://giuseppevizzari.github.io/WOA-proceedings-archive/pdfs/woa2004/18.pdf
33. Mamei, M., Zambonelli, F.: Programming stigmergic coordination with the TOTA middleware. In: Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M.J. (eds.) 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), 25–29 July 2005, Utrecht, The Netherlands, pp. 415–422. ACM (2005). https://doi.org/10.1145/1082473.1082537
34. Mamei, M., Zambonelli, F.: Programming modular robots with the TOTA middleware. In: Nakashima, H., Wellman, M.P., Weiss, G., Stone, P. (eds.) 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS

2006), Hakodate, Japan, 8–12 May 2006, pp. 485–487. ACM (2006). https://doi.org/10.1145/1160633.1160722
35. Mamei, M., Zambonelli, F., Leonardi, L.: Programming coordinated motion patterns with the TOTA middleware. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. Parallel Processing, 9th International Euro-Par Conference, Klagenfurt, Austria, 26–29 August 2003. Proceedings. Lecture Notes in Computer Science, vol. 2790, pp. 1027–1037. Springer (2003). https://doi.org/10.1007/978-3-540-45209-6_140
36. Mariani, S.: Parameter engineering vs. parameter tuning: the case of biochemical coordination in MoK. In: Baldoni et al. [3], pp. 16–23 (2013). http://ceur-ws.org/Vol-1099/paper5.pdf
37. Mariani, S., Cabri, G., Zambonelli, F.: Coordination of autonomous vehicles: taxonomy and survey. ACM Comput. Surv. (CSUR) **54**(1), 1–33 (2021)
38. Mariani, S., Omicini, A.: Space-aware coordination in ReSpecT. In: Baldoni et al. [3], pp. 1–7 (2013). http://ceur-ws.org/Vol-1099/paper3.pdf
39. Mariani, S., Omicini, A.: Tuple-based coordination of stochastic systems with uniform primitives. In: Baldoni et al. [3], pp. 8–15 (2013). http://ceur-ws.org/Vol-1099/paper4.pdf
40. Mariani, S., Omicini, A.: Coordination mechanisms for the modelling and simulation of stochastic systems: The case of uniform primitives. SCS M&S Mag. IV **4**(3), 6–25 (2014). https://hdl.handle.net/11585/480572
41. Mariani, S., Omicini, A.: TuCSoN coordination for MAS situatedness: towards a methodology. In: Santoro, C., Bergenti, F. (eds.) WOA 2014 – 15th Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 1260, pp. 48–57. Catania, Italy (2014). http://ceur-ws.org/Vol-1260/paper11.pdf
42. Mascardi, V., Omicini, A. (eds.): The Agents Journey: Twenty-five Years of Multi-agent Systems at WOA. Lecture Notes in Computer Science – State-of-the-Art Surveys. Springer (2026)
43. Menezes, R., Omicini, A., Viroli, M.: Have ReSpecT for LogOp. In: De Paoli et al. [23], pp. 94–99 (2002). http://giuseppevizzari.github.io/WOA-proceedings-archive/pdfs/woa2002/18.pdf
44. Messina, F., Pappalardo, G., Santoro, C.: A self-organising system for resource finding in large-scale computational grids. In: Omicini and Viroli [51], pp. 110–116 (2010). http://ceur-ws.org/Vol-621/paper16.pdf
45. De Mola, F., Quitadamo, R.: An agent model for future autonomic communications. In: De Paoli et al. [22], pp. 51–59 (2006). http://ceur-ws.org/Vol-204/P07.pdf
46. Montagna, S., Viroli, M., Pianini, D., Fernandez-Marquez, J.L.: Towards a comprehensive approach to spontaneous self-composition in pervasive ecosystems. In: De Paoli and Vizzari [24], pp. 89–97 (2012). http://ceur-ws.org/Vol-892/paper1.pdf
47. Nardini, E., Viroli, M., Casadei, M., Omicini, A.: A self-organising infrastructure for chemical-semantic coordination: experiments in TuCSoN. In: Omicini and Viroli [51], pp. 117–125 (2010). http://CEUR-WS.org/Vol-621/paper17.pdf
48. Omicini, A.: Formal ReSpecT in the A&A perspective. Electron. Notes Theor. Comput. Sci. **175**(2), 97–117 (2006). https://doi.org/10.1016/J.ENTCS.2007.03.006
49. Omicini, A., Denti, E.: Formal ReSpecT. Electron. Notes Theor. Comput. Sci. **48**, 179–196 (2001). https://doi.org/10.1016/S1571-0661(04)00156-2. Declarative Programming – Selected Papers from AGP 2000, La Habana, Cuba, 4–6 December 2000

50. Omicini, A., Denti, E.: From tuple spaces to tuple centres. Sci. Comput. Program. **41**(3), 277–294 (2001). https://doi.org/10.1016/S0167-6423(01)00011-9
51. Omicini, A., Viroli, M. (eds.): WOA 2010 – 11th Workshop "From Objects to Agents". CEUR Workshop Proceedings, vol. 621. Rimini, Italy (2010). http://ceur-ws.org/Vol-621/
52. Omicini, A., Zambonelli, F.: Co-ordination of mobile information agents in TuCSoN. Internet Res. **8**(5), 400–413 (1998). https://doi.org/10.1108/10662249810241266
53. Ricci, A., Omicini, A.: Agent coordination contexts: Experiments in TuCSoN. In: De Paoli et al. [23], pp. 14–21 (2002). http://giuseppevizzari.github.io/WOA-proceedings-archive/pdfs/woa2002/17.pdf
54. Rosendahl, R., Cala, A., Kirchheim, K., Lueder, A., D'Agostino, N.: Towards smart factory: multi-agent integration on industrial standards for service-oriented communication and semantic data exchange. In: Cossentino et al. [18], pp. 124–132 (2018). http://ceur-ws.org/Vol-2215/paper_20.pdf
55. Rowstron, A.I.T.: Bulk primitives in Linda run-time systems. Ph.D. thesis, University of York, UK (1996). https://rowstron.azurewebsites.net/papers/thesis_double.pdf
56. Viroli, M., Casadei, M.: Biochemical tuple spaces for self-organising coordination. In: Field, J., Vasconcelos, V.T. (eds.) Coordination Models and Languages, 11th International Conference, COORDINATION 2009, Lisboa, Portugal, 9–12 June 2009. Proceedings. Lecture Notes in Computer Science, vol. 5521, pp. 143–162. Springer (2009). https://doi.org/10.1007/978-3-642-02053-7_8
57. Viroli, M., Nardini, E., Castelli, G., Mamei, M., Zambonelli, F.: Coordinating spatially-situated pervasive service ecosystems. In: Fortino et al. [27], pp. 19–27 (2011). http://ceur-ws.org/Vol-741/ID13_ViroliNardiniCastelliMameiZambonelli.pdf
58. Viroli, M., Omicini, A.: Coordination as a service. Fundam. Inform. **73**(4), 507–534 (2006). http://content.iospress.com/articles/fundamenta-informaticae/fi73-4-04
59. Viroli, M., Ricci, A.: Timed coordination artifacts with ReSpecT. In: Baldoni et al. [9], pp. 77–85 (2004). http://giuseppevizzari.github.io/WOA-proceedings-archive/pdfs/woa2004/12.pdf
60. Wegner, P.: Why interaction is more powerful than algorithms. Commun. ACM **40**(5), 80–91 (1997). https://doi.org/10.1145/253769.253801
61. Wyckoff, P., McLaughry, S.W., Lehman, T.J., Ford, D.A.: T spaces. IBM Syst. J. **37**(3), 454–474 (1998)
62. Zambonelli, F.: Nature-inspired spatial metaphors for pervasive service ecosystems. In: Baldoni et al. [8], pp. 61–67 (2008). http://www.pa.icar.cnr.it/woa08/materiali/paper/paper_10.pdf
63. Zambonelli, F., et al.: Developing pervasive multi-agent systems with nature-inspired coordination. Pervasive Mob. Comput. **17**, 236–252 (2015). https://doi.org/10.1016/j.pmcj.2014.12.002. Special issue "10 years of Pervasive Computing" In Honor of Chatschik Bisdikian
64. Zambonelli, F., Viroli, M.: From service-oriented architectures to nature-inspired pervasive service ecosystems. In: Omicini and Viroli [51], pp. 102–109 (2010). http://ceur-ws.org/Vol-621/paper15.pdf