Multiagent System Engineering: the Coordination Viewpoint

Paolo Ciancarini* Andrea Omicini† Franco Zambonelli‡

* DSI - Università di Bologna, Mura Anteo Zamboni 7
Bologna, Italy
ciancarini@cs.unibo.it

† LIA - DEIS - Università di Bologna, Viale Risorgimento 2
Bologna, Italy
aomicini@deis.unibo.it

‡ DSI - Università di Modena, Via Campi 213b
Modena, Italy
franco.zambonelli@unimo.it

Abstract. The paper focuses on the design of multiagent systems and argues that traditional approaches fall short when dealing with complex multiagent systems. On this basis, this paper shows how an approach based on coordination models can help in the design of multiagent systems. A simple example in the area of conference management is assumed as a case study to clarify the concepts expressed.

1 Interaction in the Agent's Space

Multiagent systems are more and more becoming an ubiquitous paradigm for the design and implementation of complex software applications. Even though somehow blurred throughout the vast literature on multiagent systems, the notion of agent can be characterised by few fundamental key-points: *(i)* autonomy, *(ii)* interaction, and *(iii)* task. In other words, an agent may be thought as an *autonomous* software component which *interacts* with its environment in order to achieve its *tasks*.

Autonomy guarantees that an agent will pursue its goals proactively, so as to accomplish its tasks. Accordingly, the process of designing a multiagent system is typically conceived as picking out the multiplicity of tasks which have to be accomplished in order to achieve the intended multiagent system's goal(s), and delegating them to the agent's responsibility. In other words, the focus is mainly on intra-agent aspects, such as agent's internal structure, its view of the world, its beliefs, desires and intentions. This typically includes also delegating to agents the responsibility of coordinating itself with the other application agents accordingly to specific interaction protocols embedded into the agents [15]. Some recent proposals elect interaction to a key issue in the design of multiagent systems [25], to be tackled with from the preliminary analysis phase. Still, even in this case, agent interactions are simply considered in terms of *communication* with other agents and interaction protocols, without considering higher-level design issues related to the fact that agents will eventually live dipped into possibly complex multiagent systems.

Both the above approaches typically lead to a two-phase engineering methodology for multiagent system: first, the agents composing the systems are identified and designed according to the tasks they have been assigned; then, they are simply made interacting so as to actually build the multiagent ensemble. However, this is usually done without any explicit modelling of the global behaviour of the resulting multiagent systems. According to this (compositional) perception, a multiagent systems is viewed as the mere sum of its parts (the agents). What is relevant, then, is the role of autonomous agents acting so as to achieve their tasks: agent interaction adds nothing but the capability to compose agents into an ensemble, enabling them to work together. This corresponds to considering multiagent systems as a simple multitude of individuals, and disregarding the social aspects such as collective behaviour, social rules, and so on.

Moreover, this endorses a view of the space of agent's interaction as merely the *space of communication*, where agent-generated knowledge is made available and consumed, and where interaction histories simply results from the chaotic interleaving of the observable behaviours of single agents [23]. Correspondingly, many research efforts in the multiagent field actually deal with the problem of *enabling communication* among agents: ACLs [21] (such as KQML [9] or FIPA [10]), allowing knowledge to be transferred from one agent to another, middleware components (mediators [24], information brokers) and infrastructures (such as ORBs [13]), providing communication with some degree of transparency concerning ontology, name systems, services, and so on. However, simply focussing on enabling communication and defining agent interaction protocols does not give multiagent system designers a way to effectively deal with the complexity of the agent communication space: it makes it exist, but does not really help in governing it.

Instead, an holistic (non-compositional) view of multiagent systems is currently being promoted by several recent research efforts, introducing notions such as *social agency* [21], *socialware* [14], and *social laws* [20]. These approaches assume that the full understanding of multiagent systems calls for a comprehensive theoretical setting for *agent societies*, defining what is the world where agents live and socialise, which kinds of individuals populate the world, and how the world is ruled. Moreover, they recognise that social rules, far from being a mere limitation of the behaviour of individuals, constitutes a further source of intelligence in multiagent systems (we could call it *social intelligence*), which goes beyond the one provided by single agents, and represents a natural place where global system properties can be embodied.

This suggests that, in the context of multiagent systems, engineering the social aspects is at least as much relevant as engineering single agents. Accordingly, the social issue of a multiagent system should be accounted for since the earliest design phases, and social rules and collective behaviour should be the subjects of a separate design phase, independent from the design of individual agents.

2 Coordination as a Design Dimension

Shifting focus from the agent internal structure to the agent interaction space changes the way in which a multiagent system is designed. The design process of a multiagent

system can still be conceived as a two-phases process, but of a quite different sort. In the first phase, a multiagent system has to be analysed and understood so as identify both the *individual tasks*, to be carried out by single agents, and the *social tasks*, to be achieved by groups of agents as the result of their mutual interaction. In the second phase, individual tasks drive the design and implementation of individual agents. Social tasks, instead, should drive the design and implementation of the agent interaction space, that is to say, both the agent interaction protocols and the agent interaction rules.

For instance, consider the case of a multiagent system in charge of managing the review process for an international conference [3]. An individual task for an agent representing a PC member would be picking up ten papers for each referee. A social task, instead, would be ensuring that the agents representing the PC members globally select, for each submitted paper, at least three referees each. Here, the individual task could be easily and naturally charged upon a single agent representing a referee. Instead, the social task could be more naturally pursued by imposing a social rule, that is, by ruling the interaction space somehow.

In the example above and in general, a frequently adopted solution is delegating social tasks to especially-designed agents. However, in our opinion, exploiting agents – which are built and interact with other agents in the same way as any other agent – in order to rule agent interaction seems not a clean and elegant approach. Instead, rather than a solution, this turns out to be a clear symptom of the lack of abstractions and tools specifically suited to govern the agent interaction space.

In order to fully support the engineering of multiagent systems, then, something more is needed than theories and tools for designing and implementing agents, and languages and middleware for enabling interagent communication. While agent languages and architectures may satisfy the need for single agent design, enabling communication may not suffice: in order to design societies, enforce social rules, and embodying collective behaviour, the key-point is *ruling communication*. From an engineering viewpoint, this raises the question of how social rules and collective behaviour can be built, and embedded into multiagent systems. Since societies of any kind are grounded on the interaction among the single components, it seems natural to identify the space of agent interaction as the place where social rules and policies have to be represented and embodied. If the interaction space is recognised as an independent space for multiagent system design, abstractions and tools are needed which enable agent interaction to be shaped independently of the individual agents.

Managing the interaction is the goal of *coordination* [22]: the space of agent interaction is no longer to be seen as merely the space of communication, but also the *space of coordination*. *Coordination languages*, *models* and *architectures* [2] provide for abstractions and tools to manage and rule the agent interaction space, in the same way as agent languages and architectures provide for abstractions and tools to build up agents. In short, coordination languages are meant to express the agent's observable behaviour and to design its interaction protocol; coordination models allow the interaction space to be shaped and ruled, while coordination architectures provide for patterns for the organisation of agent ensembles. All in one, coordination theory is what is needed to fully support the engineering of multiagent systems, particularly in the design and implementation of social tasks.

3 Coordination Models and Languages

In its general terms, coordination is the art of managing interactions and dependencies among activities, that is, among agents, in the context of multiagent systems [2, 12]. A coordination model provides a formal framework in which the interaction of software agents can be expressed. Generally speaking, a coordination model deals with agent creation and destruction, agent communication activities, agent distribution and mobility in space, as well as the synchronisation and distribution of agent actions over time. A coordination model can be thought as consisting of three elements:

- the *coordinables*, which are the entities whose mutual interaction is ruled by the model – agents, in multiagent systems;
- the *coordination media*, which are the abstractions enabling agent interactions, as well as the core around which the components of a coordinated system are organised. Examples are semaphores, monitors, channels, or more complex media like tuple spaces, blackboards, etc.
- the *coordination laws*, which define the behaviour of the coordination media in response to interaction events. The laws can be defined in terms of a *communication language*, that is a syntax used to express and exchange data structures, and a *coordination language*, that is, a set of interaction primitives and their semantics.

From a software engineering viewpoint, a coordination model works as a source for design metaphors, abstractions, and mechanisms effectively supporting the definition of the software architecture and the development process of a multicomponent software system.

3.1 Classes of Coordination Models

According to [19], coordination models can be divided in two classes: *data-driven* and *control-driven* ones. In control-driven coordination models, like MANIFOLD [18], coordinables (agents) typically open themselves to the external world and interact with it through events occurring on well-defined input/output ports. The observable behaviour of the coordinables, from the point of view of the coordination media, is the one of state changes and events occurring on these ports. The coordination laws establish how events and state changes can occur and how they propagate. Therefore, the coordination media basically handle the topology of the interaction space, with no concern for the data exchanged between coordinables.

In data-driven coordination models, like Linda [11], coordinables interact with the external world by exchanging data structures through the coordination media, which basically act as shared data spaces. The coordination laws determine how data structures are represented and how they are stored, accessed, and consumed. Unlike control-driven coordination models, the coordination medium has no perception of the changes in the state of coordinables, and does not provide any control over the communication topology.

The choice of the model for agent coordination has indeed an impact on the design of a multiagent system. As argued in [4], data-driven models seems to better suit

open systems, where a number of possibly a-priori unknown and autonomous entities have to cooperate, as typical in the case of multiagent systems. In this case, focussing on dependencies between the components, as a control-driven model would do, would somehow clash with the autonomy of the agents and the unpredictability of the open environment. Focussing on the knowledge exchanged among agents, besides better coping with agent autonomy and unpredictability, seems to better suit intelligent multiagent systems, where agents are typically designed to deal with information rather than with control. However, the engineering of multiagent systems actually requires more control on the interaction space than the one provided by data-driven models. This obviously calls for new, hybrid models of coordination, combining the focus on information typical of data-driven models with the flexibility and control provided by control-driven ones.

3.2 Hybrid Coordination Models

A typical problem of data-driven coordinated systems is the built-in and fixed behaviour of the shared dataspace used as the coordination medium: neither new communication primitives can be added, nor can new behaviour be defined in response to standard communication events. As a result, either the mechanisms provided by the coordination model for accessing the shared dataspaces are powerful enough to express the coordination laws required to effectively govern agent ensembles, or agents have to be charged in their code with the burden of social rules.

To address this deficiency, *hybrid* coordination models are meant to combine the cleanness and elegance of data-driven models with the flexibility and power of control-driven ones. On the one hand, communication still occurs through the exchange of data structure via a shared dataspace. On the other hand, the shared dataspace can act not only as a communication channel, but also as a *programmable coordination medium* [5], capable of detecting any communication event and associating any required behaviour to it.

An example of a hybrid coordination model is endorsed by the notion of *tuple centre*, introduced in [6], and currently exploited by several Internet-oriented coordination models [8, 1, 17] as their coordination medium. More precisely, a tuple centre is a communication abstraction which is perceived by the interacting entities as a standard Linda-like tuple space, but whose behaviour in response to communication events can be defined so as to embed the laws of coordination. So, while a Linda tuple space supports only those coordination policies that can be directly mapped onto its fixed behaviour, the control over the interaction space provided by a tuple centre can be exploited in order to bridge between different representations of information shared by agents, to provide new coordination mechanisms, to support the full monitoring of agent interaction, and, above all, to embed the laws for agent coordination [6].

Further examples of hybrid models are T Spaces [26] and Law-governed Linda [16]. Law-governed Linda exploits programmable tuple spaces for security and efficiency in distributed systems, where programmable controllers are associated locally to agents to intercept and modify the behaviour of communication operations. In T Spaces, agents can add new primitives or overload old ones in order to implement any kind of transaction they need on the data stored in the tuple space.

Altogether, all these models endorse a view of coordination where agents communicate by exchanging information in form of tuples, in a data-driven fashion, but coordination is achieved by properly detecting and managing communication event, in a control-driven fashion. Agent interaction protocols can be designed as simply and expressively as in data-driven models, whereas the agent interaction space can be easily managed by exploiting the underlying powerful control-driven mechanisms. As a result, hybrid coordination abstractions, like tuple centres, seem well-suited to be used as the core for the engineering of agent societies in multiagent systems [7].

4 A Case Study

In this section, we sketch the proof-of-concept implementation of a multiagent application. The goal is to show how an approach based on coordination impacts on the the design and development of multiagent applications, as well as on their maintenance.

Let us consider the process related to the production of the proceedings for an international conference [3]. The process consists in several phases, each involving several people. During the preliminary submission phase, potential authors have to submit their papers, and be acknowledged that papers print correctly and have been assigned a submission number. Once the submission deadline is expired, the PC committee has to handle the review of the papers, contacting potential referees and asking them to review some of the papers. After a while, reviews are expected to come in and be used to decide about the acceptance/rejection of the submissions. Authors must be notified of the decisions and, in case of acceptance, must be asked to produce the camera ready version of their revised papers. The publisher has to collect the camera ready versions and eventually print the whole proceedings.

As sketched above, the process is essentially a workflow one: the activities of a number of different people have to be somehow synchronised, data have to be exchanged, and each person involved may be in need of performing different activities depending on the global status of the process. Clearly, the whole process can be automatised by modelling it in terms of a multiagent system. The most obvious design choice is to associate an agent to each of the sub-tasks of the process. An agent associated to a given task has to complete it both by exploiting its own “intelligence”, and by interacting with the person that is actually in charge of the task (e.g., an agent cannot review a paper, but it can help the reviewer in filling the review form and delivering it in time).

In order to better point out the coordination issues which may arise in this application, we concentrate now on the review process only – disregarding submission and final publication details – where several different coordination rules can be devised, depending on how the PC committee handles the process.

Let us consider first a small conference where only PC members act as reviewers and the PC chair decides which PC member has to review which papers – each PC member is represented by a “PC Member Agent” (PCMA), and the PC chair by a “PC Chair Agent” (PCCA). Given the number of submissions, and the number of PC members, the individual task of the PC chair (and of the PCCA, correspondingly) is to set the number of required reviews for every submitted paper, and, by consequence, the number of reviews to be provided by each PC member/reviewer. This, it turn, defines

the individual task of each PC member (and of each PCMA), that is, to review as many papers as required. What has to be determined is who/what is put in charge of of ensuring that each paper actually receives the required number of different reviews from different PC members – which is obviously a social task for the multiagent system.

A traditional agent-oriented methodology would obviously make an agent take care of the social task, too – an *ad-hoc* agent, possibly, or, in this case, the PCCA itself, more likely. In fact, the PCCA can have a global view of the whole process, and can easily check whether all PC members receive the correct number of papers to review, and whether all papers receive enough reviews. This would require that well-defined interaction protocols are defined between the PCCA and the PCMAs in order to negotiate the review of a paper, and eventually send them both the papers and the review form, and that the PCCA keeps track of all the paper assignments, explicitly driving the selection process towards the accomplishment of the global goal by the agents' ensemble. Even though this approach may seem quite simple and clean, we claim it is not. On the one hand, the PCCA is charged of a task which is not an individual one, but rather a social one. On the other hand, the solution is neither easily scalable, nor incrementally refinable, since any change to the specifications is likely to force the modification of all the agents involved (more specifically, of their interaction protocols).

In principle, adopting a pure control-driven coordination model for the application would results in a very similar design, where a "coordinator" component would be in charge of topologically connecting the PCCA with the proper PCMAs according to the global needs for paper's reviews. However, the emphasis on control induced by control-driven models does not easily cope with the decentralisation of control promoted by autonomy of coordinables, i.e., the agents, as argued in [4].

Instead, when adopting a data-driven coordination model, the PCCA can insert papers and review forms as appropriate data structures into the dataspace, and the PCMAs can look on the dataspace for papers to review However, the achievement of the social task of ensuring a fair number of different reviewers to each paper requires again the definition of interaction protocols between the agents and the shared dataspace: the PCCA may for instance provide papers in the dataspace as resources to be consumed by PCMAs – when all the occurrences of the same paper in the dataspace have been consumed, it is ensured to receive as many reviews as required. This also requires that PCMAs agree on a protocol which consumes papers from the dataspace (and does not simply read them), and takes care of never getting the same paper more than once.

When adopting an hybrid coordination model, the already simple design of the application is likely to be further simplified or, at least, made cleaner. For instance, a tuple centres could be easily exploited as the core for the MAS' social task, by embodying the required coordination laws. So, the PCCA has simply to put into the tuple centres the papers and the review forms, whereas the PCMAs has simply to retrieve papers from the tuple centre. In its turn, the tuple centre is in charge of monitoring the accesses to the dataspace, and can be programmed to ensure that all papers receives the required number of reviews, for instance making visible to PCMAs selecting papers only those ones which have not already been ensured of the minimum number of reviews.

Let us now complicate a little bit the example, by supposing that the dimension of the conference explodes, in terms of amount of submitted papers, so as to make

it unfeasible the above sketched centralised management. In this case, while each PC member still has to take care of a number of papers, he is no longer in charge of revising it by him/herself, but has instead to find an appropriate reviewer for each paper of his/her, external to the PC. In this scenario, in addition to ensuring that each paper is assigned to as many different PCMAs as are the required reviews, one has also to ensure that two different PCMAs do not choose the same referee for the same paper. This can be considered as an further social task to be pursued by the MAS.

The traditional agent-oriented approach sketched above would require a redesign of the PCCA (which is in charge of one more task) – its representation of the world, its interaction protocol – and of the PCMAs, too, since not only paper assignment to PC members, but also the referee's choice has now to be negotiated. This lack of incremental refinability is basically due to the fact that agent coordination is put in charge of an agent – whatever this means from a conceptual viewpoint – instead of a coordination medium. A social task is charged upon an individual agent, which basically says that this approach does not provide abstractions suitable for the design of agent societies.

The adoption of a hybrid coordination model, instead, makes it possible to keep neatly distinct individual and social tasks, by encapsulating the coordination laws (i.e., the social rules) into a separate coordination component, e.g., a tuple centre. In this case, in fact, no change would be required to the PCCA. In turn, the PCMAs would be simply required to slightly change their interaction protocol, by associating a reviewer's name to each request for a paper (e.g., trying to consume a tuple with both a paper and a reviewer, instead of a tuple with a paper only). The tuple centre should then be programmed so as to satisfy a PCMA request only after checking the proposed paper-reviewer association (e.g., by either returning or refusing the paper-reviewer tuple), thus encapsulating most of the changes in terms of new rules governing agent communication – which seems quite appropriate, after all. As a result, both PCCA and PCMAs can be simply designed around their individual tasks, and social tasks are in charge of the coordination medium, embodying social laws in terms of rules over agent communication – so that, for instance, a change to social tasks does not imply a re-design of the whole system.

For examples of code, and more details about programming tuple centres for the coordination of multiagent systems, we forward the interested reader to [17].

5 Conclusions

Traditional approaches to multiagent system design fall short in handling the complexity related to the management of highly interactive multiagent applications. In fact, in most of the cases, the rules governing the interactions between the application agents are not explicitly modelled at the earlier design stages, and the design usually mixes computation and coordination into the code of the agents, without being able to define and separate individual and social application tasks. This is likely to undermine the openness and the reusability of applications, as well as their maintainability.

In this paper we have tried to point out some relevant consequences of adopting a coordination viewpoint on the engineering of a multiagent system, which are all basically

related to the *separation of concerns* between the computation and the coordination issues, or, in other terms, between individual and social tasks:

- *design focus* – agent interaction protocols can be designed by focussing on its individual task(s) (e.g., concentrating on the information needed and produced/inferred by the agent in the process of achieving its tasks), in many sense disregarding the social task(s) of the groups the agent belongs to;
- *coordination design* – social rules and collective behaviour can be represented in terms of coordination laws, charged upon the coordination media, and can be designed and implemented independently, with no concern for the internal structure of the agents.
- *social intelligence* – social tasks can be achieved as the result of the mutual interaction of agents, each one pursuing its own aims. The interaction space is then a further source of intelligence for the multiagent system
- *modularity and reusability* – autonomous agents focussing on their tasks, with no concern for coordination issues and designed around very straightforward interaction protocols, are a natural source of modularity for multiagent systems. They can be reused wherever their capabilities are needed, independently of the social rules. Dually, coordination rules can be exploited to achieve social tasks simply given the agent goals and protocols, independently of the internal structure of the agents.
- *incremental design and implementation* – once that social rules are designed and implemented, individual capabilities can be refined so as to improve the agent ability to achieve its task independently of the rest of the system. Analogously, once that agents tasks and interaction protocols are designed and implemented, coordination rules can be independently refined so as to improve the capability of the multiagent system to achieve its social goals.

References

1. Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. MARS: a programmable coordination architecture for mobile agents. *IEEE Internet Computing*. To appear.
2. Paolo Ciancarini. Coordination models and languages as software integrators. *ACM Computing Surveys*, 28(2), June 1996.
3. Paolo Ciancarini, Oscar Nierstrasz, and Robert Tolksdorf. A case study in coordination: Conference management on the Internet. `http://www.cs.unibo.it/~cianca/wwwpages/case.ps.gz`.
4. Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli. Coordination technologies for Internet agents. *Nordic Journal of Computing*. To appear.
5. Enrico Denti, Antonio Natali, and Andrea Omicini. Programmable coordination media. In David Garlan and Daniel Le Métayer, editors, *Coordination Languages and Models – Proceedings of the 2nd International Conference (COORDINATION'97)*, volume 1282 of *LNCS*, pages 274–288, Berlin (D), September 1–3 1997. Springer-Verlag.
6. Enrico Denti, Antonio Natali, and Andrea Omicini. On the expressive power of a language for programming coordination media. In *Proceedings of the 1998 ACM Symposium on Applied Computing (SAC'98)*, pages 169–177, Atlanta (GA), February 27 - March 1 1998. ACM. Track on Coordination Models, Languages and Applications.

7. Enrico Denti and Andrea Omicini. Designing multi-agent systems around an extensible communication abstraction. In Amedeo Cesta and Pierre-Yves Schobbens, editors, *Proceedings of the 4th ModelAge Workshop on Formal Models of Agents (ModelAge'97)*, pages 87–97, Certosa di Pontignano (I), January 15–18 1997. National Research Council of Italy. To be published by Springer-Verlag in the LNAI Series.
8. Enrico Denti and Andrea Omicini. An architecture for tuple-based coordination of multi-agent systems. *Software — Practice & Experience*, 29(12):1103–1121, October 1999.
9. Timothy W. Finin, Richard Fritzson, Donald McKay, and Robin McEntire. KQML as an agent communication language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg (Maryland), November 1994. ACM Press.
10. Foundation for Intelligent Physical Agents. FIPA'99. `http://www.fipa.org`.
11. David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.
12. David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.
13. Object Management Group. CORBA 2.1 specifications, 1997. `http://www.omg.org`.
14. Fumio Hattori, Takeshi Ohguro, Makoto Yokoo, Shigeo Matsubara, and Sen Yoshida. Socialware: Multiagent systems for supporting network communities. *Communications of the ACM*, 42(3):55–61, March 1999. Special Section on Multiagent Systems on the Net.
15. Carlos Iglesias, Mercedes Garijo, and José Gonzales. A survey of agent-oriented methodologies. In Jörg P. Müller, Munindar P. Singh, and Anand S. Rao, editors, *Intelligent Agents V — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555 of *Lecture Notes in Artificial Intelligence*, pages 317–330. Springer-Verlag, Berlin, 1999.
16. Naftaly H. Minsky and Jerrold Leichter. Law-governed Linda as a coordination model. In Paolo Ciancarini and Oscar Nierstrasz, editors, *Object-Based Models and Languages for Concurrent Systems – Proceedings of the ECOOP'94 Workshop on Models and Languages for Coordination of Parallelism and Distribution*, volume 924 of *Lecture Notes in Computer Science*, pages 125–146. Springer-Verlag, Berlin, 1994.
17. Andrea Omicini and Franco Zambonelli. Coordination for Internet application development. *Journal of Autonomous Agents and Multi-Agent Systems*, 2(3), September 1999. Special Issue on Coordination Mechanisms and Patterns for Web Agents.
18. George A. Papadopoulos. Distributed and parallel systems engineering in MANIFOLD. *Parallel Computing*, 24(7):1137–1160, 1998.
19. George A. Papadopoulos and Farhad Arbab. Coordination models and languages. *Advances in Computers*, 46:The Engineering of Large Systems:329–400, August 1998.
20. Joav Shoham and Moshe Tennenholtz. Social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73, 1995.
21. Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):55–61, December 1998.
22. Peter Wegner. Coordination as constrained interaction. In Paolo Ciancarini and Chris Hankin, editors, *Coordination Languages and Models – Proceedings of the 1st International Conference (COORDINATION'96)*, volume 1061 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, Berlin, Cesena (I), April 15–17 1996.
23. Peter Wegner. Why interaction is more powerful than computing. *Communications of the ACM*, 40(5):80–91, May 1997.
24. Gio Wiederhold. Mediators in the architecture of future information system. *IEEE Computer*, 25(3):38–49, March 1992.

25. Michael J. Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents*, pages 69–76. ACM, Seattle (WA), May, 1–5 1999.
26. Peter Wyckoff, Stephen W. McLaughry, Tobin J. Lehman, and Daniel A. Ford. T Spaces. *IBM Journal of Research and Development*, 37(3 - Java Techonology):454–474, 1998.