Objective versus Subjective Coordination in the Engineering of Agent Systems

Andrea Omicini[1] and Sascha Ossowski[2]

[1] DEIS, Università di Bologna a Cesena
Via Rasi e Spinelli 176, 47023, Cesena, FC, Italy
aomicini@deis.unibo.it http://lia.deis.unibo.it/~ao
[2] ESCET, Universidad Rey Juan Carlos
Campus de Mostoles, Calle Tulipan s/n, E-28933 Madrid, Spain
S.Ossowski@escet.urjc.es http://www.ia.escet.urjc.es/~sossowski

Abstract. The governance of interaction is a critical issue in the engineering of agent systems. Research on *coordination* addresses this issue by providing a wide range of models, abstractions and technologies. It is often the case, however, that such a wide range of proposals could not easily find an unitary and coherent conceptual framework where all the different views and solutions can be understood and compared – and this is particularly true in the context of agent models and systems.
In this paper, we first discuss how all the many diverse approaches to agent coordination can be categorised in two main classes – the *subjective* and *objective* approaches – , depending on whether they adopt the agent's or the engineer's viewpoint, respectively. We then claim that the two approaches have a deep and different impact on the way in which agent systems are modelled and built, and show two examples rooted in different models and technologies. Finally, we advocate that both approaches play a fundamental role in the engineering of agent systems, and that any methodology for the design and development of agent systems has to exploit both objective and subjective coordination models and technologies.

1 Introduction

Multiagent systems (MAS henceforth) are software systems made up of multiple encapsulated computational entities, which are embedded in an environment and (inter-)act in an autonomous and intelligent fashion. When exactly software modules, objects, components etc. can be called agents, and as to how far and in what sense they are to behave intelligently and autonomously, is still subject to considerable debate. However, it is commonly accepted that coordination is a key characteristic of MAS and that, in turn, the capability of coordinating with others constitutes a centrepiece of agenthood.

Still, when researchers are asked to nail down their intuitive notion of what coordination in MAS is all about, agreement usually ends. To some respect this is not surprising, as the term is used in a variety of disciplines, such as

M. Klusch et al. (Eds.): Intelligent Information Agents, LNAI 2586, pp. 179–202, 2003.

Economy, Sociology and Biology, each with its own epistemological apparatus and research agenda. To make things worse, even within Computer Science the term is used quite differently in fields like robotics, concurrent programming, and mainstream software engineering. As a result, many different techniques and frameworks coexist in the agent community that claim to tackle the problem of coordination in MAS in one or another way, a situation that tends to be confusing not only to novice MAS designers. We feel that a clarification in this respect is urgently needed if, as we foresee, the development of heterogeneous and distributed systems based on the agent metaphor are to become next "big thing" in software engineering.

In this chapter, we first discuss how all the many diverse approaches to agent coordination can be categorised in two main classes, *subjective* and *objective* approaches, depending on whether they adopt the agent's or the engineer's viewpoint, respectively. We argue that the deep and different impact of these approaches on the way in which agent systems are modelled and built can provide important clues for a successful navigation through the jungle of the many, seemingly unrelated, works on agent coordination. Moreover, we advocate that both approaches play a fundamental role in the engineering of MAS for open environments such as the Internet, and that any modern methodology for the design and development of such agent-based applications will have to exploit both objective and subjective coordination models and technologies.

This chapter is organised as follows: we first review some of the most relevant coordination models, abstractions and technologies and relate them to each other. Subsequently, we introduce the notions of subjective and objective coordination, illustrate them by an example, and discuss their impact on the design of coordination mechanisms for MAS. Section 4 describes our position on how these notions may impact coordination related issues in agent-oriented software engineering. This chapter is concluded by our vision of the lines along which research on coordination in MAS may evolve.

2 Coordination: An Overview

This section provides a brief overview of current coordination models, abstractions and technologies. We first outline different attempts to *conceptualise* coordination in MAS. Then, we compare these abstractions along a variety of dimensions and discuss the role of "coordination middleware": software infrastructures that support the *instrumentation* of different coordination models.

2.1 Models of Coordination

Maybe the most widely accepted conceptualisation of coordination in the MAS field originates from Organisational Science. It defines coordination as *the management of dependencies* between organisational activities [11]. One of the many workflows in an organisation, for instance, may involve a secretary writing a letter, an official signing it, and another employee sending it to its final destination.

The interrelation among these activities is modelled as a *producer/consumer* dependency, which can be managed by inserting additional *notification* and *transportation* actions into the workflow.

It is straightforward to generalise this approach to coordination problems in multiagent systems. Obviously, the subjects whose activities need to be coordinated (sometimes called *coordinables*) are the agents. The entities between which dependencies arise (or: *objects of coordination*) are often termed quite differently, but usually come down to entities like goals, actions and plans. Depending on the characteristics of the MAS environment, a taxonomy of dependencies can be established, and a set of potential coordination actions assigned to each of them (e.g. [29]). Within this model, the *process* of coordination is to accomplish two major tasks: first, a *detection* of dependencies needs to be performed, and second, a *decision* respecting which coordination action to apply must be taken. A coordination *mechanism* shapes the way that agents perform these tasks [18].

The *result* of coordination, and its *quality*, is conceived differently at different levels of granularity. Von Martial's stance on coordination as *a way of adapting to the environment* [29], is quite well suited to understand this question from a *micro-level* (agent-centric) perspective, in particular if we are concerned with multiagent settings. If new acquaintances enter an agent's environment, coordination amounts to re-assessing its former goals, plans and actions, so as to account for the new (potential) dependencies between itself and other agents. If a STRIPS-like planning agent, for instance, is put into a multiagent environment, it will definitely have to accommodate its individual plans to the new dependencies between its own prospective actions and potential actions of others, trying to exploit possible synergies (others may free certain relevant blocks for it), and avoiding harmful dependencies (making sure that others do not unstack intentionally constructed stacks etc). At this level, the result of coordination, the agent's adapted individual plan, is the better the closer it takes the agent to the achievement of its goals in the multiagent environment.

From a *macro-level* (MAS-centric) perspective, the outcome of coordination can be conceived a "global" plan (or decision, action etc.). This may be a "joint plan" [23], if the agents reach an explicit agreement on it during the coordination process, or just the sum of the agents' individual plans (or decisions, actions etc. – sometimes called "multi-plan" [18]) as perceived by an external observer. Roughly speaking, the quality of the outcome of coordination at the macro-level can be evaluated with respect to the agents' joint goals or the desired functionality of the MAS as a whole. If no such notion can be ascribed to the MAS, other, more basic features can be used instead. A good result of coordination, for instance, is often supposed to be efficiency [23], which frequently comes down to the notion of Pareto-optimality: no agent could have increased the degree of achievements of its goals, without any other being worse off in that sense. The amount of resources necessary for coordination (e.g. the number of messages necessary) is also sometimes used as a measure of efficiency.

The dependency model of coordination appears to be particularly well suited to *represent* relevant features of a coordination problem in MAS. The TÆMS

framework [5], for instance, has been used to model coordination requirements in a variety of interesting MAS domains. It is also useful to rationalise observed coordination behaviour in line with Newell's knowledge-level perspective [13]. Still, when designing coordination processes for real-world MAS, things are not as simple as the dependency model may suggest. Dependency detection may come to be a rather knowledge intensive task, which is further complicated by incomplete and potentially inconsistent local views of the agents. Moreover, making timely decisions that lead to efficient coordination actions is also everything but trivial [10]. The problem becomes even more difficult when agents pursuing partially conflicting goals come into play [18]. In all but the most simple MAS, the instrumentation of these tasks gives rise to complex patterns of interactions among agents. The set of possible interactions is often called the *interaction space* of coordination.

From a *software engineering* perspective, coordination is probably best conceived as the effort of *governing the space of interaction* [1] of a MAS. When approaching coordination from a *design* stance, the basic challenge amounts to how to make agents converge on interaction patterns that adequately (i.e. instrumentally with respect to desired MAS features) solve the dependency detection and decision tasks. A variety of approaches to tackle this problem can be found in the literature. Multiagent planning, negotiation, organisational structures, conventions, norms, reputation management, and mechanism design, are just some of them. These approaches aim at shaping the interaction space either directly, by making assumptions on agent behaviours and/or knowledge, or indirectly, by modifying the *context* of the agents in the MAS environment. The applicability of these mechanisms depends largely on the characteristics of the coordination problem at hand, as we will outline in the next section.

Finally, let us insist that conceiving coordination as a means to rule and manage interaction in MAS is by no means opposing the dependency model. Rather, it shifts the focus of attention from representation to design. We will elaborate further on this software engineering perspective on coordination in Section 4.

2.2 Characteristics of Coordination Models

It is a commonplace that, the more open a multiagent environment, the more difficult it is to instill a sufficient quality of coordination (see also the chapter by Fredriksson, Gustavsson, and Ricci in this volume [7]). For instance, in a closed environment, as assumed traditionally by Distributed Problem-Solving Systems, agent behaviour is controlled at *design-time*. As the agent designer has full control over the agents, she can implement a coordination mechanism of her choice: if certain assumptions on the agents' behaviours are necessary, these can simply be "hard-coded" into the agent programs. A popular example is the original Contract-Net Protocol (CNP) [25]: volunteering services to a contractor relies on the assumption of benevolence from the side of the bidders, which can be easily achieved when agents are designed to follow the CNP. Thus, the space

of interactions is completely determined at the time the MAS is built. In closed environments, design choices are usually driven by efficiency issues.

At the other extreme, in large-scale open networks like the Internet, agent behaviour is *uncontrolled*, so that very few assumptions can be made about agents' behaviours and their frequencies. In particular, it is almost impossible to globally foresee and to influence the space of potential agent interactions. Most probably, agents will behave in a self-interested fashion, which may help to anticipate some of their actions, and may provide some clues on how to design coordination strategies at the micro-level. Also, in certain "parts" of the open system it may be possible to influence the frequencies of behaviours by spawning new agents with desired characteristics, so as to improve the quality of coordination at the macro-level (e.g. [21]). Still, very little can be done in the general case, so that the main focus of design comes to be placed on other issues, such as security, and often at a lower level of analysis.

A promising approach to efficiently instill coordination in open systems is inspired by the notion of agent as a situated entity. It falls in between the two extremes (fully controlled vs. uncontrolled), by providing for a sort of *partially controlled* agent behaviour. Since the environment where agents and MAS live is partially under human control, agent interaction can be influenced by engineering the agent environment: in particular, agent *infrastructures* are typically exploited to shape the agent environment according to the engineers' needs. For instance, *coordination infrastructures* provide agents with coordination (run-time) abstractions embodying *coordination as a service* [28], that exert a form of partial *run-time control* over agent behaviour. Coordination infrastructures are not meant to influence agent behaviour directly, but they can affect agent actions. As a trivial example, a security infrastructure enforces a set of behavioural restrictions for potential users that implicitly bounds the admissible agent interaction histories: the effects of actions of agents on the environment are thereby constrained without limiting their deliberation capabilities. Given a range of different *coordination services* made available by an infrastructure, agents can freely choose a service based on their self-interest: once they register with a coordination service, however, the infrastructure will enforce compliance with behavioural restrictions. This may be achieved by executing mobile agents on specific virtual machines, or by making agents interact through "intelligent" communication channels that govern the interaction space by filtering, modifying, or generating certain messages. Future agent-based auctions may become examples of such coordination services.

This latter form of coordination is often termed *mediated coordination* – which in general could rely on either a distinguished middle agent [9] or a coordination abstraction provided by an infrastructure (like a Linda tuple space [8], or a ReSpecT tuple centre [16]). In the literature, mediated coordination is often confused with *centralised coordination*. In fact, another important dimension of coordination models amounts to whether they can be designed in a *centralised* fashion, or need a *decentralised* instrumentation. While centralised mechanisms are appropriate for closed environments with design-time coordi-

nation, decentralised mechanisms (like peer-to-peer models) better satisfy the needs of open environments with run-time coordination. However, mediated coordination is often *multi-centric* – i.e., neither centralised, nor fully decentralised –, thus achieving a sort of welcome compromise between the engineering urges (pushing toward controlled and predictable systems) and typical features of the systems of today (emphasising openness, dynamics, and unpredictability).

Closely related to the above discussion is a concept recently introduced by Tolksdorf [27]: A coordination mechanism is said to be *coupled* if the effectiveness of an agent's coordination behaviour is based on assumptions on some (other) agent's behaviour. By contrast, *uncoupled* mechanisms impose no assumptions on agent behaviour. As truly decentralised coordination can only be achieved by a coupled mechanism, it bears the additional burden of ensuring that all involved agents will behave as expected.

Finally, when shifting our attention to the micro-level, the distinction between *quantitative* and *qualitative* models of coordination comes into play [18]. Qualitative approaches basically follow the dependency model outlined earlier, by directly representing the different "reasons" for preferring or not certain objects of coordination to others. An agent's coordination behaviour is guided by whether a certain local action (plan, goal, etc.) depends positively or negatively on the actions of others: it will choose its local and communicative actions based on the "structure" of dependencies that it shares with its acquaintances. So, in cooperative environments, it is straightforward to conceive coordination as a kind of *constraint satisfaction problem* [18]. In quantitative models, by contrast, the structure of the coordination problem is hidden in the shape of a multi-attribute utility function. An agent has control over only some of the function's attributes (i.e. some of the objects of coordination), and its utility may increase (decrease) in case there is a positive (negative) dependency with an attribute governed by another agent, but these dependencies are not explicitly modelled. Its local coordination decision problem then corresponds to a special type of *optimisation problem*: to determine a local action (plan, goal, etc.), and to induce others to choose local actions (plans, goals, etc.), so as to maximise its local utility. The quantitative approach may draw upon a well developed theoretical framework for both, cooperative settings (Operations Research) and non-cooperative settings (Game Theory), but suffers from that fact that, due to the uncertainties intrinsic to MAS domains, the utility function is only an approximation, so that its optimum need not coincide with an agent's actual best choice in the real environment. On the other hand, a coordination mechanism based on qualitative dependencies is less prone to such modelling inaccuracies, but its foundations go back to theories from social sciences (e.g. Social Psychology), that do not provide a sound formal framework to guide local decision-making.

3 Subjective *versus* Objective Coordination

Subjective and objective coordination are two different but complementary ways of conceiving coordination in MAS. We first sketch the different perspectives on

coordination that give rise to these notions, and then provide an example for each of them, relying on a simple, closed environment for illustrative purposes. Finally, we discuss why both approaches are needed for the engineering of open agent systems.

3.1 Different Perspectives on Interaction

Basically, there are two ways to look at interaction: from the inside and from the outside of the interacting entities. When taking MAS into account, this comes to say that interaction within a MAS can be seen from the viewpoints of either an agent, or an external observer not directly involved in the interaction: respectively, the *subjective* and the *objective* viewpoints [24]. From the subjective viewpoint of an agent, the space of interaction roughly amounts to the observable behaviour of other agents and the evolution of the environment over time, filtered and interpreted according to the individual agent's perception and understanding. From the objective viewpoint, the space of interaction is basically given by the observable behaviour of all the agents of a MAS and the agent environment as well, and by all their *interaction histories* [30]. If coordination is taken to mean governing the interaction, then the two different viewpoints lead to two different ways of coordinating.

When looking at interaction from the agent's own point of view, coordination roughly amounts to *(i)* monitoring all interactions that are perceivable and relevant to the agent, as well as their evolution over time, and *(ii)* devising actions that could bring the overall state of the MAS (or, more generally, of the world) to (better) coincide with one of the agent's own goals. In short, this is what is called *subjective coordination*. So, in general, the acts of an agent coordinating within a MAS are driven by its own perception and understanding of the other agents' behaviour, capabilities and goals, as well as of the environment state and dynamics.

On the other hand, when looking at interaction within a MAS from outside the interacting agents, as an external observer, coordination means affecting agent interaction so as to make the resulting MAS evolution accomplish one or more of the observer's goals.[1] In short, this is what is called *objective coordination*. In general, the acts of external observers – whether they be MAS designers, developers, users, managers, or even agents working at the meta-level – are influenced not only by their perception and understanding of MAS agents and environment, but also by their a-priori knowledge of the agents' aims, capabilities and behaviour. Also, some forms of understanding and foreseeing of the global behaviour of the MAS as the result of the overall interaction amongst the agents and the environment are required to instill a coordination that is effective over time from the standpoint of the observer.

[1] Affecting agent interaction does not mean (directly) affecting agents. That is why objective coordination by no means contrasts with the fundamental notion of agent *autonomy* [32].

3.2 Subjective vs. Objective Coordination: An Example

This section aims at illustrating the subjective and objective viewpoints, using a reactive coordination scenario as an example. This scenario involves cognitive agents that develop short-term plans. Such a stance is appropriate for a variety of real world domains, such as agent-based decision support [4][20]. In the following, we will use a simple setting in the synchronous blocks domain [18], an extension of the well known blocks world, for illustrative purposes.

There are a table of unlimited size and four numbered *blocks*. Blocks can be placed either directly on the table or on top of another block, and there is no limit to the height of a stack of blocks. The only operations that can be performed therefore are to place a block x on the table (formally: $move(x, table)$), which requires x to be clear, or to place a block x on top of some other block y (formally: $move(x, y)$), which additionally requires y to be also clear. There is a *clock* that marks each instant of time by a tick. This scenario is synchronous, since all operations occur at a given tick, and the clock is the same for all operations. So, any operation could in principle be labelled by its tick, and any set of operations is totally ordered, since any operation can be said to occur before, after, or at the same time (tick) as any other operation. A *plan* of length k is a sequence of k operations performed successively. Instead of an operation, a plan may contain a *NOP*, indicating that nothing is done at a certain tick.

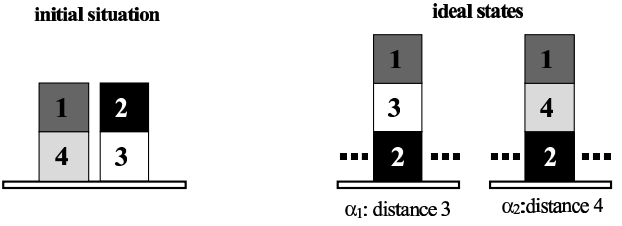

Fig. 1. A scenario in the synchronous blocks domain

Consider the scenario shown in Fig. 1. There are two agents, α_1 and α_2, both looking only two actions ahead, *i.e.* generating plans of length 2. In addition, suppose that the former is capable of moving all blocks but block 4, while the latter may manipulate all blocks but block 3: so, a plan by α_1 cannot include operations of the form $move(4, y)$, while a plan by α_2 cannot include operations of the form $move(3, y)$.

The initial situation and the states that each agent would ideally like to bring about are shown in Fig. 1. Both agents measure the "distance" to their ideal world states by the minimum number of actions they would need to perform in order to reach it. Table 1 shows some individual plans; "executability" denotes an agent's ability to enact a specific plan.

The subjective viewpoint in the example. In a single-agent environment, the agent's subjective attitude is straightforward: among the executable plans

Table 1. Some individual plans

plan	operation sequence	ability	executability
π_1	$[move(2, table), move(3, 2)]$	α_1	true
π_3	$[move(2, table), NOP]$	α_1, α_2	true
π_4	$[move(1, table), NOP]$	α_1, α_2	true
π_9	$[move(2, table), move(4, 2)]$	α_2	false
π_{10}	$[move(1, 2), move(4, 1)]$	α_2	true
π_{11}	$[move(2, 1), move(3, 2)]$	α_1	true
π_ε	$[NOP, NOP]$	α_1, α_2	true

that it can enact, it will choose the one that takes it as close as possible to its ideal state. Still, in the example α_1 and α_2 act in a shared environment, and may perform their plans simultaneously, i.e. in essence they perform a so-called *multiplan* [18]. Suppose plans are compositional, that is, the result of a multiplan μ be the "sum" of the effects of its component plans. If the component plans of a multiplan "interfere", the following rules apply:

- A multiplan μ is *not executable* if some component plans access the same block at the same tick in different ways, or if they obstruct a block that another component plan uses at a later tick.
- A multiplan μ is *executable* despite a non-executable component plan, if other component plans "complement" it, e.g. by providing a missing action.

Let us denote with (π_i,π_j) the multiplan where π_i and π_j are performed simultaneously.

Then, the multiplans (π_1,π_4), (π_4,π_9) and (π_3,π_4), for instance, lead to the world states shown in Fig. 2, respectively.

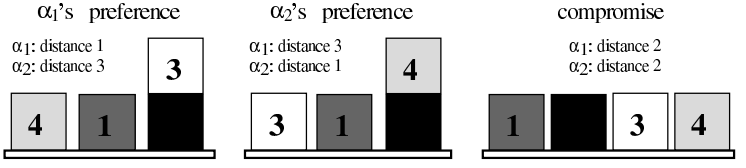

Fig. 2. Some outcomes of the execution of multiplans

Each agent just controls one component of the multiplan that is performed, so they have a (mutual) interest in coordinating their choices. It is then appropriate that they exchange a sequence of messages following some negotiation protocol, which defines the space of possible interactions for coordination. Obviously, the specific coordination interaction that actually occurs depends on the local choices of the agents when making offers and counteroffers respecting the multiplans to agree on. Supposing that agents are individually rational, these choices should depend on the agents' potential to influence each other, i.e. how far they may manipulate the outcome of their acquaintances' individual plans.

In the example, three classes of such social dependencies can be identified. First, there is a *feasibility-dependence* of agent α for a plan π with respect to α' if the latter can invalidate the plan, i.e. if it can turn down the execution of π. In the example, each agent is in a feasibility-dependence to the other for all plans shown in Table 1 except π_ϵ. Second, agent α is *negatively dependent* for a plan π with respect to α', if α' can deviate the outcome of π to a state that is less preferred by α. If α' can bring about a change in the outcome of α's plan π that α welcomes, then α is *positively dependent* on α'. In Table 1, each agent depends positively on the other for π_3 and π_4.

So, in essence, designing a coordination mechanism from a subjective point of view means

- to design a social reasoning mechanism that detects these dependence relations
- to endow agents with a decision algorithm that, taking into account the different dependence relations, guides the agent's decision making during coordination interactions (negotiation).

For instance, a social reasoning mechanism could make α_1 understand both negative and positive dependencies for multiplans with respect to α_2. Also, a decision algorithm could lead α_1 and α_2 to choose a multiplan like (π_3,π_4), producing a compromise between the ideal states of α_1 and α_2 (Fig. 2). Both represent forms of subjective coordination, where the agents reason, plan and act in order to make the global effect of agent interaction achieve their desired state of the world.

The objective viewpoint in the example. Suppose now that we are the designers of the global MAS in the synchronous blocks world. We are looking at the agents interactions from the outside, with the aim of obtaining certain global results (i.e. prefer certain classes of multiplans to others). From such an objective stance on the coordination, we need to tackle two aspects of coordination:

- a model of the outcome of coordination in the present multiagent environment is needed. What is the present space of interactions, and what will be potential instances of interactions and their outcomes?
- a mechanism to modify the space of interactions in a principled way is required. How can we "shape" the space of interactions and promote/hamper certain types of interactions, so as to influence the outcome of coordination in a desired direction?

So, based on the expected outcome of agents coordinating from their subjective viewpoints, and our mechanisms to affect the agents' interaction space and its global effects from an objective point of view, we can "tune" coordination from an objective stance.[2]

[2] Obviously, this is not to say that the objective stance always prevails over the subjective point of view. In fact, in open environments a change in objective coordination laws (e.g. tax laws) will in turn have repercussions in the agents' subjective coordination strategies (e.g. the tax payers' behaviour).

In the synchronous blocks world domain example, the first aspect is best tackled from within a *quantitative* framework. Suppose that for an agent the utility of a multiplan is given by the reduction of distance to the agent's ideal state. The utilities that each agent obtains from a multiplan can be represented in a vector. In the example, the utility vectors of multiplans (π_1,π_4), (π_4,π_9) and (π_3,π_4) for α_1 and α_2 are (2,1), (0,3) and (1,2), respectively. Agents may agree to gamble over multiplans, i.e. decide to "flip a coin" in order to choose between alternative courses of action. In this case we use the expected utility, i.e. the sum of each multiplan utility weighed by its probability. So, the utility of a mixed multiplan that enacts the above three multiplans with equal probability would be (1,2).

Still, agents are autonomous, so they are not forced to reach an agreement. Their interaction for mutual coordination may as well end up in conflict, which is equivalent to the decision not to (explicitly) coordinate their actions, and instead take the chances individually. In this case, an agent must account for the unpleasant eventuality that all its acquaintances could jointly try to harm it. So, a rational agent may choose the best individual plan that is "autonomous", i.e. whose executability cannot be turned down by others. For instance, the only plan that α_1 can execute and which is guaranteed not to become incompatible is π_ε, which α_2 counters by π_{10}, resulting in a conflict utility of -2 for α_1. Agent α_2 also needs to choose π_ε in case of disagreement, to which α_1's most malicious response is to enact π_{11}, giving rise to a conflict utility of -1 for α_2. So, the conflict utility vector in our example is $\boldsymbol{d} = (-2, -1)$.

The set S of utilities of all feasible (mixed) multiplans, along with the disagreement point, describes the so-called *bargaining scenario* [26]. In our two-agents example, the bargaining scenario can be represented in a plane, where each axis measures the utility of one agent (Fig. 3).

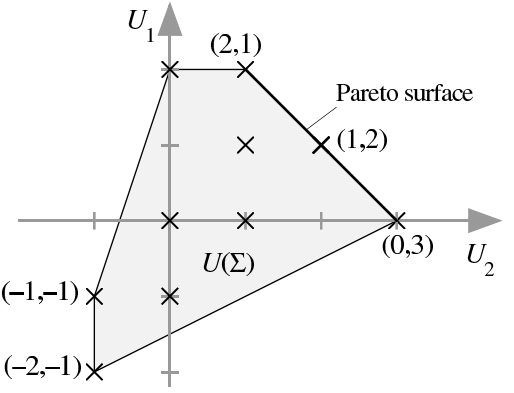

Fig. 3. Graphical representation of the example scenario

In order to model the outcome of subjective coordination among agents from a global point of view, the whole mathematical apparatus of bargaining theory

is now applicable. In particular, we can apply the Nash bargaining solution [12] which states that rational agents should agree on a multiplan that maximises the product of gains from the disagreement point. In our example, this leads to a solution vector of (1,2). Consequently, our model predicts that the outcome of subjective coordination is a (mixed) multiplan whose utility is given by this solution, e.g. an agreement on the "compromise" multiplan (π_3,π_4).

Still, from a designer's point of view, the outcome of coordination-oriented interactions that are driven entirely by the subjective interests of the involved agents does not necessarily lead to the desired characteristics and functionalities of the MAS as a whole. As a consequence, a mechanism for objective coordination is required in order to suitably bias the coordination process and its outcome toward the achievement of the global MAS goals.

For instance, a simple objective coordination technique in the synchronous blocks domain relies on the ability to issue permissions or prohibitions for certain agents to perform specific actions, and assumes that agents (cannot but) comply with these prescriptions. Suppose it is forbidden for all agents to put block 1 on block 2. As a result, agent α_2 can no longer execute plan π_{10}, whereas α_1 is permitted to enact π_4. So, the worst situation that α_2 can bring about in the eyes of α_1 is to put block 4 on top of 2. The conflict utility of α_2 remains unchanged, so that the disagreement point changes in favour of α_1 to $\boldsymbol{d} = (-1, -1)$. Because of the change in $\boldsymbol{d}$, the outcome of subjective coordination interaction will switch to $(1.5, 1.5)$, which is reached by randomising equally between "compromise" (π_3,π_4) and α_1's favourite (π_1,π_4). If it was permitted for agent α_2 to unstack block 1 at the first tick (i.e. all plans that manipulate the left stack in the first step are forbidden for α_1), the worst thing that α_1 can still do is to obstruct block 2 for α_2 by putting block 4 on it in the second step, giving rise to a disagreement point which moves towards α_2. The solution utility vector then changes to $(0.5, 2.5)$, which is reached by selecting α_2's favourite (π_4,π_9) and compromise (π_3,π_4) both with probability $p = 0.5$.

Notice that the outcome of coordination has been changed without prescribing agents exactly what to do: it is just the space of possible agreements that has been modified. Still, this is not of much help as long as we do not have a means to estimate the effects of such changes in the interaction space. We would like to ensure that, as in the examples, if an agent is favoured/harmed by prescriptions and its "negotiation position" is strengthened/weakened, then this should be reflected in the agreement reached. In fact, this is always the case given the above mechanism. Issuing permissions and prohibitions shifts the disagreement point towards the favoured agents, while the property of *disagreement point monotonicity* of the Nash bargaining solution ensures that the corresponding change in the coordination outcome correlates with the movement [26]. The outcome of objective coordination can actually be computed based on a *coupled* approach using a distributed constraint optimisation algorithm [19].

3.3 Impacts on the Engineering of Agent Systems

As illustrated by the previous example, roughly speaking, subjective coordination affects the way in which individual agents behave and interact, whereas

objective coordination affects the way in which interaction amongst the agent and the environment is enabled and ruled. So, while the main focus of subjective coordination is the behaviour of agents as (social) *individuals* immersed in a MAS, the emphasis of objective coordination lies more on the behaviour of a MAS as a whole.

When defining the architecture and the inner dynamics of individual agents, the subjective viewpoint on coordination is clearly the most pertinent one. How other agents' actions are represented and foreseen, how to interpret and handle shared information in the agent system, when and why to move from an agent environment to another, and so on – all these issues concern subjective coordination, and affect the way in which the agents of a MAS are designed, developed and deployed as individual entities. For instance, agents α_1 and α_2 in the example should be designed to have some planning ability, to share goals, and to be able to understand mutual dependencies to make it possible for them to reach at least the compromise outcome of Fig. 2. So, the viability of approaches adopting a subjective coordination viewpoint to the engineering of MAS strictly depends not only on the mental (reasoning, planning and deliberation) capabilities of the agents, but also on their ability to foresee the effect of their actions on the environment, the behaviour of the other agents, and the overall dynamics of the environment as well.

On the other hand, since in principle an external observer does not directly interact with the agents of a MAS, some capability to act on the space of MAS interaction without dealing directly with agents is obviously required in order to enable any form of objective coordination. Given that agents are typically situated entities, acting on the agent environment makes it possible to affect the behaviour of an agent system without having to alter the agents themselves. Objective coordination therefore deals with the agent environment: modifying the virtual machine supporting agent functioning, changing resource availability and access policies, altering the behaviour of the agent communication channel, be it virtual or physical, and so on – all these are possible ways to influence and possibly harness the behaviour of a MAS without directly intervening on individual agents. The viability of objective coordination in the engineering of agent systems depends then on the availability of suitable models of the agent environment, and on their proper embodiment within agent infrastructures. There, objective coordination would conceivably take on the form of a collection of suitably expressive coordination abstractions, provided as run-time coordination services by the agent infrastructure. In order to be able to formulate and ensure global properties of a MAS, the behaviour of coordination abstractions in response to events in the agent interaction space should be well-known and predictable.

To this end, the agents in the example could either willingly accomplish the prescriptions about blocks, or e.g. be forced to do so by a suitable infrastructure implementing security policies. For instance, blocks might embody security mechanisms, and directly handle permissions, thus bounding the space of admissible interactions, or, in other terms, shaping the interaction space of the agents. Or, blocks may be accessible only by means of virtual arms provided by the hosting infrastructure, and moves would be provided as services by the infras-

tructure itself. Permissions would be handled by the infrastructure, that would then implement what we call *coordination as a service*, that is, coordination provided as a service to agents by the infrastructure through a run-time coordination abstraction [28]. In both cases, agent coordination would be influenced by something external to the agents.

Also, it might be the case that permissions are explicitly available to the agent's inspection. Rules governing access to resources (blocks, in the example) might be explicitly represented and stored so as to be inspectable to agents. Agents may then read permissions, understand them, and possibly plan their course of action accordingly, thus obtaining a clear benefit by their increased knowledge of the environment resources. Inspectability of coordination rules (permissions, in the example) is where subjective and objective coordination meet. Roughly speaking, explicit representation of coordination laws, externally to agents, is an objective form of coordination, while their interiorisation and use by agents is obviously a form of subjective coordination. Intuitively, then, this example shows the benefits of blending together both subjective and objective coordination in the engineering of MAS.

On the one side, in fact, a purely subjective approach to coordination in the engineering of agent systems would endorse a mere reductionistic view, coming to say that agent systems are compositional, and their behaviour is nothing more than the sum of the individual's behaviour – an easily defeasible argument, indeed. Among the many consequences, this would require global properties of the agent system to be "distributed" amongst individuals, providing neither abstractions nor mechanisms to encapsulate such properties. As a result, the purely subjective approach would directly entail lack of support for design, development, and, even more, deployment of agent systems' global properties – which would result in substantial difficulties for incremental development, impractical run-time modification, and so on. On the other side, a purely objective approach to coordination in the engineering of agent systems would endorse a rough holistic view – where only inter-agent dependencies and interactions count, and individuals' behaviour has no relevance for global system behaviour. Among the many consequences, this would stand in stark contrast with any notion of agent autonomy, and would prevent agents from featuring any ability to affect the environment for their own individual purposes – no space for anything resembling an agent left, in short.

In the end, all the above considerations suggest that any principled approach to the engineering of agent systems should necessarily provide support for both subjective and objective models of coordination, possibly integrating them in a coherent conceptual framework, and providing at the same time a suitable support for all the phases of the engineering processes – in terms of coordination languages, development tools, and run-time environments. According to that, in the next section we discuss the role of coordination models (both subjective and objectives) in the context of agent-oriented methodologies for the engineering of software systems.

4 Coordination in Agent-Oriented Software Engineering

Coordination is a key issue for any Agent-Oriented Software Engineering methodology (AOSE). The notions of objective and subjective coordination may come to play a crucial role in this respect, in particular if we follow the notion of coordination as a service. In the sequel, we outline some basic concepts of AOSE and discuss their relation to coordination services and their supporting infrastructures. We then show how subjective and objective coordination models are put to use in the engineering of a MAS for the domain of workflow management.

4.1 Agent Abstractions in System Engineering

The ever-growing complexity of modern systems has raised the requirements on engineering models and processes up to an unprecedented level: new abstractions and practices are required to deal with open, heterogeneous, distributed and highly dynamic systems. In this perspective, Agent-Oriented Software Engineering is a new research area meant to exploit powerful agent-based metaphors and abstractions in the engineering of complex software systems [3,33]. Notions like *agent*, *agent society*, and *agent environment* constitute the basis for many of the AOSE methodologies currently being developed, such as Gaia [34] or SODA [14]. With respect to current best practice – often based on object-oriented technologies and methodologies – agent-based approaches feature two main properties: the autonomy of the main components (agents are autonomous entities encapsulating control), and the promotion of interaction to a first-class issue (agents are typically social and situated entities).

Autonomy of agents typically takes on two forms, at different levels of abstractions: at the higher level, agents are assigned to *tasks* that they pursue in an autonomous way; at the lower level, agents are autonomous and independent *loci* of control. Task accomplishment drives the flow of control inside the agents, so that a task can be regarded as the high-level abstraction to manage the low-level mechanisms of control. By moving the focus of system design up from *control dependencies* to *task dependencies*, autonomy of agents is then what allows to abstract away from control. Since one of the main sources of complexity in the design of open, distributed, highly dynamic systems is the coupling of control flows between different components, autonomy plays a key role in making agents suitable abstractions for the engineering of complex systems.

Agents are also typically acknowledged to be *social* entities, coexisting with other agents in a MAS and interacting with them in order to accomplish their own task. A group of agents interacting within a MAS is often referred to as a *society*, in particularly when their mutual interaction and communication is headed (either intentionally or not) toward the achievement of a global (*social*) functionality. From this viewpoint, then, societies of agents also serve as powerful design abstractions, to be handled as first-class entities throughout the whole engineering process. In particular, societies can be charged with tasks that could not (for either theoretical or practical reasons) be assigned to individual agents, which are then to be achieved by the overall coordinated activity of the individual member agents [14].

Finally, agents are *situated* entities, immersed in an environment containing not only other agents, but also resources and services. Resources might range from basic survival (such as an agent virtual machine) and inter-operability technologies, to complex coordination middleware. In fact, the activity of any agent (and of any agent society as well) could not be thought of or modelled without considering the environment wherein it lives – so that the agent environment calls for suitable first-class engineering abstractions on its own, modelling how each agent perceives the world around, and how it can predict its evolution over time.

Both task assignment to agents and interaction with the environment are typically mediated and represented by the notion of *role*. In fact, tasks are typically assigned to roles, that are assumed by agents either statically at design time, or dynamically at run-time. When playing a role, an agent is in charge of the corresponding task, and is entitled of all the authorisations and permissions (and limitations as well) pertaining to its role. Roles typically determine which part is played by agents within a MAS interpreted as an *organisation*, so they might be hierarchically related, or generally define some dependency between agents. In general, the interactions between the different roles have to follow specific rules for the overall organisation to work correctly and efficiently toward a shared global goal.

In all, the engineering process as promoted by agent-oriented abstractions is basically *task-oriented*: individual tasks are assigned to agents, and drive the design and development of individual agents, social tasks are assigned to groups of agents (societies) organised so as to pursue a common goal. In principle, we no longer delegate control to components (as in object-oriented and component-based practice), instead we delegate tasks, and with them, *responsibility* – while agents and societies encapsulate control. As a result, the design of agents and MAS is driven by tasks: individual tasks drive the design of individual agents, whereas social tasks drive the design of agent interaction protocols and supra-agent (social) rules.

4.2 Coordination in the Engineering of Agent System

Both autonomy and interactiveness – as novel features of agent-oriented software engineering – raise coordination issues. On the one hand, the ability of autonomously pursuing the accomplishment of a task requires agents to be able at least to represent the environment and foresee its evolution, to model and somehow predict the behaviour of the other agents and the effects of their own actions as well. This is of course the domain of subjective coordination approaches, as discussed in the previous sections. On the other hand, agent interaction within a society or with the environment asks to be first enabled, and then governed, in order to make the global agent system behave as required. These are actually interoperability and coordination issues, respectively, that straightforwardly fall under the umbrella of objective coordination models and technologies.

It is then quite natural here to emphasise the role that coordination models and technologies are going to play in the engineering of agent systems. This is particularly evident when considering the engineering of agent societies. Societies

can be modelled as the sum of the individual components plus the social rules, or norms, that govern their behaviour and interaction. Once social norms are interpreted as coordination laws, coordination media can be easily acknowledged as social abstractions, around which agent societies can be modelled and built. As a result, agent societies can be easily thought of as the sum of individual agents and coordination abstractions, encapsulating norms in terms of coordination laws.

So, designing an agent society basically amounts to defining the social task, designing the individual and social roles to be assigned to individual agents as well as the corresponding interaction protocols, and then defining the norms ruling the collective behaviour. Subsequently, as argued in [14], suitable models, patterns, and mechanisms, of coordination can be chosen, that are expressive enough to capture the defined norms. For instance, in a dynamic setting, both the agents and the rules are likely to change over time: agents may join and leave the society at will, rules may vary to account for a new goal to achieve, or for a change in the environmental conditions: correspondingly, coordination abstractions should support the dynamic modification of coordination laws.

For a principled approach to the construction of agent systems, design abstractions should be supported throughout the whole engineering process, from design to development and deployment. In particular, when continuous development and incremental refinement are mandatory, design abstractions should be straightforwardly mapped upon run-time abstractions. This first calls for the availability of specialised coordination-specific IDEs to monitor and inspect the space of agent interaction, equipping developers with tools mapping abstractions into manageable metaphors – in particular, social ones [6]. Then, suitable coordination infrastructures are required, providing coordination abstractions as run-time reification of agent societies, and supporting the chosen model of coordination for social norm representation and enforcement. For instance, dynamic modification of coordination laws should obviously not only be foreseen by the model, but also suitably supported at execution time.

4.3 A Case Study: Agent Coordination in Workflow Management

A typical example where both subjective and objective coordination models and technologies come at hand is workflow management. In [22], an example of Workflow Management System (WfMS) in a Virtual Enterprise (VE) is presented and discussed in detail: workflow is first shown to be amenable for an interpretation as an agent coordination problem, then a WfMS is built by exploiting the TuCSoN coordination technology. In the following, we summarise the example as well as the main issues it raises, showing how the adoption of suitable models and technologies for agent coordination can make the engineering of complex systems easier, particularly when they promote the fruitful coexistence of both subjective and objective approaches.

The virtual bookshop. In [22], a *virtual bookshop* is presented, that is, a VE that aggregates several companies of different sorts to sell books through

the Internet. In this scenario, four sorts of companies were identified: the *bookseller* (who provides the books), the *carrier* (who delivers books from sellers to customers), the *interbank service* (which executes payment transactions), and the *Internet service provider* (the Web portal for customers). In the example, two workflows were defined, that imply the specification and execution of classic workflow rules, such as managing a sequence of activities, an AND-Splitting process (coordinating activities started at the same time and executed in parallel) and an AND-Joining process (requiring synchronisation of parallel activities):

- the purchase of a single book from a specific bookseller, and
- the purchase of a set of books from different booksellers.

The first workflow describes the sequence of activities involved in the purchase of a specific book. First, order information is gathered from the Web site used by the customer – any of the sites provided by any Internet service provider participating in the VE. Then, the activity to get the book ready at the chosen bookseller's starts. When the book is ready, the dispatching activity is executed, to have the book delivered from the bookshop to the customer by a carrier. Finally, after the book has been delivered to the customer, the payment transaction activity is executed, involving the intervention of an interbank service to transfer money from the customer to the VE.

The second workflow involves the purchase of a set of books from (possibly) different booksellers. After order information is gathered, a number of book acquiring activities are executed in parallel, each aiming at getting a book from a specific bookseller. When all the books are ready at the involved booksellers', the dispatching activity can start as in the first case (for the sake of simplicity, the same carrier is assumed to be used to deliver all books).

WfMS as an agent system. The main element of a workflow is the *task*, as the elementary unit of work inside a workflow. A workflow *schema* is the collection of tasks that collectively achieve the goal(s) of a *process*. Tasks are interrelated into a flow structure via *connectors*, such as split and join elements, which define the execution dependencies among tasks (the order in which tasks can be executed). A *case* is an execution of a workflow schema, i.e. an instance of the corresponding (business) process.

According to the standard WfMC approach [31], workflow management is modelled in terms of *workflow engines* (workflow servers or coordinators, where coordination takes place) and *workflow participants* – thus straightforwardly inducing a clear separation between coordinating and coordinated activities. Workflow tasks are executed autonomously by (artificial or human) agents, coordinated by the workflow engines: in particular, agents can be used to represent the workflow participants whose activity can be automatised. Each workflow is then modelled as an agent society, involving individual agents autonomously pursuing workflow tasks, while the society, built around the workflow engine, aims at bringing the whole workflow to a successful end.

In the example, the main agents involved in the workflows are:

- *interface agents*, responsible for collecting information about customers and orders. These agents also interact with customers during order execution, informing them about order status and possible problems.
- *buyer agents*, responsible for ordering books from the booksellers and getting them ready for delivery.
- *delivering agents*, responsible for delivering the books to the customers, provided that the books are ready at the bookseller's. To do so, they inform the carrier of each new dispatch to do, and monitor the delivery until the book is in the customer's hands.
- *payment agents*, responsible for getting the payments done. These agents interact with an interbank service for transferring money from customers to the virtual bookshop.

Workflow engines govern the activity of the workflow participants, encapsulating the workflow rules. So, coordination media can be used as workflow engines to automatise coordination, embodying the workflow rules in terms of coordination laws: in this way, the workflow rules can be encapsulated outside agents and somehow superimposed on them, so that agents are entitled to autonomously pursue their task with no need to be aware of the global workflow. In particular, in the example, logic *tuple centres* are used as coordination media [16], that is, tuple spaces whose behaviour in response to communication events can be specified through ReSpecT logic programs – so that a ReSpecT specification of a tuple centre actually defines the laws of coordination that the tuple centre embodies [15].

The agent/engine interaction occurs either when an agent declares to be ready to execute a specific task (according to its role), or when an agent returns the result of the task execution (or communicates the related problems). In our example, interaction takes the form of insertion and retrieval of tuples in `vbs` tuple centres, exploiting Linda-like coordination primitives [8]: agents check for new tasks to execute by trying to retrieve (by means of an `inp` operation) `ready_to_do` tuples, and provide task results by inserting (by means of an `out`) `task_success` or `task_failure` tuples.

Mapping a workflow engine onto a tuple centre essentially means to define the event model and the process model [2]. The event model defines what kinds of internal and external workflow events must be considered. Internal events are the ones defining the normal flow of activities within the workflow, such as temporal events related to the starting and the termination of a task. External events are events that can have an impact on the regular evolution of a running case, but are outside the control of the WfMS. In the example, both internal and external events are captured and represented by *reifying* them as descriptive tuples in the tuple centre. For instance, when a case `CaseID` is terminated, the tuple `case_done(CaseID)` is inserted in the tuple centre.

In turn, the process model describes the behavioural aspects of a workflow specification, from its initial state to one of its final states. In our example, the coordination activities of a workflow engine are naturally mapped upon the behaviour of a tuple centre. In particular, part of the rules, implemented

Table 2. ReSpecT code for the workflow meta-rules

```
reaction(out(case_to_start(CaseName, Info)),(
  in_r(case_to_start(CaseName, Info)),
  in_r(case_id_counter(ID)),
  NextID is ID + 1,
  out_r(case_id_counter(NextID)),
  out_r(case_started(ID,CaseName, Info)),
  out_r(case_state(ID, executing, _)),
  out_r(next_task(ID, CaseName, case_start,_)))).

reaction(inp(ready_to_do(TaskType, _, _)),(
  pre,
  in_r(task_to_do(CaseID, TaskType, Info)),
  in_r(task_id_counter(TaskID)),
  NextID is TaskID + 1,
  out_r(task_id_counter(NextID)),
  current_agent(ExecutorID),
  out_r(ready_to_do(TaskType, Info, TaskID)),
  out_r(task_started(TaskID, CaseID, TaskType, Info, ExecutorID)))).

reaction(out(task_success(TaskID, Info)),(
  in_r(task_success(TaskID, Info)),
  out_r(task_done(TaskID, Info)))).
reaction(out_r(task_done(TaskID, Info)),(
  rd_r(task_started(TaskID, CaseID, TaskType, Info, _)),
  rd_r(case_started(CaseID, CaseName, _)),
  rd_r(case_state(CaseID, executing, _)),
  out_r(next_task(CaseID, CaseName, TaskType, Info)))).
reaction(out_r(next_task(CaseID, CaseName, TaskType, Info)),(
  in_r(next_task(CaseID, CaseName, TaskType, Info)))).

reaction(out_r(case_done(CaseID)),(
  in_r(case_state(CaseID, executing, _)))).
```

as ReSpecT reactions, constitute a basic core of meta-workflow rules, acting as the instructions of a *workflow virtual machine*: they are independent of the specific workflow, and are used to define standard workflow behaviour triggered by internal workflow events, making it possible to define the basic skeleton of the flow structure. These (meta-)rules interpret/execute other rules (trigger other ReSpecT reactions in our case), which are specific to the given workflow type and constitute the remaining part of the rules defining the process model. Taken from [22], Table 2 shows the ReSpecT code for the workflow meta-rules, while Table 3 shows the ReSpecT code for the single book purchase workflow.

Features. Adopting coordination media like tuple centres as the workflow engines provides, in the first place, all the benefits of objective coordination approaches, and, in the second place, a most effective support for subjective coordination techniques.

Due to mediated interaction, workflow participants are loosely coupled, and may interact with no need to know each other, or to coexist in space or in time. Then, workflow rules are encapsulated within suitable abstractions outside the agents – the tuple centres –, so that workflow participants are enabled to autonomously participate in the workflow with no need to be aware of the global workflow rules or dynamics. This also means that agents acting as workflow par-

Table 3. ReSpecT code for the single book purchase workflow example

```
reaction(out_r(next_task(CaseID, single_book_purchase, case_start,_)),(
  rd_r(case_started(CaseID,single_book_purchase,info(_,Book,Seller,_))),
  out_r(task_to_do(CaseID, buy_the_book, info(Book,Seller))))).

reaction(out_r(next_task(CaseID, single_book_purchase, buy_the_book,
                         BookReceipt)),(
  rd_r(case_started(CaseID,single_book_purchase,info(Customer,_,Seller,Carrier))),
  out_r(task_to_do(CaseID, dispatch_the_book,
                   info(BookReceipt,Carrier,Seller,Customer))))).

reaction(out_r(next_task(CaseID, single_book_purchase, dispatch_the_book,
                         CustomerReceipt)),(
  rd_r(case_started(CaseID,single_book_purchase,info(Customer,Book,_,_))),
  out_r(task_to_do(CaseID, make_payment, info(Customer,Book))))).

reaction(out_r(next_task(CaseID, single_book_purchase, make_payment,
                         BankReceipt)),(
  out_r(case_done(CaseID)))).
```

ticipants are amenable to independent design and development, and, dually, that changes in the workflow rules do not necessarily require changes to all agents. Finally, both the state of the interaction and the laws of coordination are explicitly represented as logic tuples in the tuple centre (respectively, ordinary tuples, and meta-level ReSpecT tuples), that can then be inspected and dynamically modified by both engineers and (intelligent) agents, at least in principle.

On the other hand, explicit representation and inspectability of both the communication state and the coordination state are the features of objective coordination models (like ReSpecT tuple centres) that build a bridge toward subjective approaches, thereby making them particularly effective. In fact, since agents can perceive the state of the interaction, an intelligent cooperative agent could in principle monitor the state of the workflow and act to improve the overall workflow performance or results, either starting an activity by itself, or by properly stimulating other cooperative (but possibly not-so-intelligent) agents. Also, since coordination laws are inspectable and dynamically modifiable, an intelligent supervisor agent could in principle reason about the state of the workflow and its rules, and possibly change them according to current needs – for instance, when external conditions have turned the previously established workflow ineffective – thus paving the way for dynamically adaptive systems.

All in all, this example clearly suggests that the concerted exploitation of both objective and subjective approaches in the coordination of agent systems could provide a well defined path to the construction of dynamic, adaptive, and intelligent systems.

5 Discussion

In this chapter, we have given a brief overview over current conceptualisations, models and support infrastructures for coordination in MAS. Among the many dimensions of coordination models, the duality between the objective and the

subjective viewpoint on coordination has been identified as particularly important for the construction of modern MAS, an idea that has been illustrated using a simple multiagent blocks world domain as an example. We have stressed the fact that such an approach leads quite naturally to the idea of conceiving coordination as a service that multiagent coordination infrastructures may provide. Finally, we have argued that this stance can be smoothly integrated into current Agent Oriented Software Engineering methodologies, and provided evidence for that claim by an example from the domain of agent-based workflow management.

The question of how to design effective coordination mechanisms for open environments is a major research issue in the agent community. A popular approach to this problem is to complement mechanisms for closed systems with techniques that enforce initially uncontrolled agents to comply with the required "behavioural norms". Some self-enforcing approaches have been borrowed from Game Theory and Economics (e.g. Vickrey Auctions), but their applicability to many real-world domains appears to be limited. In this respect, work on social control and models of trust is promising, but much work still needs to be done in that direction. An effective instrumentation of findings from Evolutionary Game Theory and Social Simulation to instill desired global properties, e.g. by controlling the frequencies of certain behaviours in a open agent society, is still further ahead.

At the time being, the notion of *coordination as a service* appears most promising to us. Once agents have freely chosen a particular coordination service, the compliance with its "behavioural norm" can be enforced by the infrastructure of the service itself, e.g. through dedicated virtual machines or "intelligent" communication channels. Still, this idea requires advances on several fronts. Most importantly, more powerful service description languages are needed, that allow agents to dynamically choose between services based on their self-interest. In addition, what kind of coordination knowledge should be used in these services, and how to model, represent, and enact it within coordination media is still subject to debate. Finally, some considerations on how current AOSE methodologies could be extended in a principled manner to better capture the design abstraction of coordination services and its mapping to adequate run-time abstractions may lead to interesting results. In this regard, the chapter by Fredriksson, Gustavsson, and Ricci in the present volume [7] presents a current effort to develop a comprehensive methodology for the provision of sustainable coordination in open systems.

References

1. Nadia Busi, Paolo Ciancarini, Roberto Gorrieri, and Gianluigi Zavattaro. Coordination models: A guided tour. In Omicini et al. [17], chapter 1, pages 6–24.
2. Fabio Casati, Silvana Castano, Mariagrazia Fugini, Isabel Mirabel, and Barbara Pernici. Using patterns to design rules in workflows. *IEEE Transactions on Software Engineering*, 26(8):760–785, August 2000.
3. Paolo Ciancarini and Michael J. Wooldridge, editors. *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes on Computer Science*. Springer-Verlag, January 2001. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000, Revised Papers.

4. José Cuena and Sascha Ossowski. Distributed models for decision support. In Gerhard Weiss, editor, *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*, pages 459–504. MIT Press, 1999.
5. Keith Decker. TÆMS: A framework for environment centered analysis and design of coordination mechanisms. In Gregory M.P. O'Hare and Nicholas R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 429–448. John Wiley and Sons, 1996.
6. Enrico Denti, Andrea Omicini, and Alessandro Ricci. Coordination tools for MAS development and deployment. *Applied Artificial Intelligence*, 16(10), November 2002.
7. Martin Fredriksson, Rune Gustavsson, and Alessandro Ricci. Sustainable coordination. Springer-Verlag, 2002. In this volume.
8. David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
9. Matthias Klusch and Katia Sycara. Brokering and matchmaking for coordination of agent societies: A survey. In Omicini et al. [17], chapter 8, pages 197–224.
10. Victor R. Lesser. Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Autonomous Agents and Multi-Agent Systems*, 1(1):89–111, 1998.
11. Thomas Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.
12. John F. Nash. The bargaining problem. *Econometrica*, 28:152–155, 1950.
13. Allen Newell. Reflections on the knowledge level. *Artificial Intelligence*, 59:31—38, 1993.
14. Andrea Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In Ciancarini and Wooldridge [3], pages 185–193. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000, Revised Papers.
15. Andrea Omicini and Enrico Denti. Formal ReSpecT. In Agostino Dovier, Maria Chiara Meo, and Andrea Omicini, editors, *Declarative Programming – Selected Papers from AGP'00*, volume 48 of *Electronic Notes in Theoretical Computer Science*, pages 179–196. Elsevier Science B. V., 2001.
16. Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, November 2001.
17. Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer-Verlag, March 2001.
18. Sascha Ossowski. *Coordination in Artificial Agent Societies – Social Structure and its Implications for Autonomus Problem-solving Agents*, volume 1535 of *LNAI*. Springer, Berlin, 1999.
19. Sascha Ossowski. Constraint-based coordination of autonomous agents. *Electronic Notes in Theoretical Computer Science*, 48, 2001.
20. Sascha Ossowski, Josefa Z. Hernández, Carlos A. Iglesias, and Alberto Fernández. Engineering agent systems for decision support. In Paolo Petta, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in an Agent World III*. Springer-Verlag, 2002.
21. Sunju Park, Edmund H. Durfee, and William P. Birmingham. Emergent properties of a market-based digital library with strategic agents. *Autonomous Agents and Multiagent Systems*, 5:33–51, 2000.

22. Alessandro Ricci, Andrea Omicini, and Enrico Denti. Virtual enterprises and workflow management as agent coordination issues. *International Journal of Cooperative Information Systems*, 11(3/4):355–380, September/December 2002. Cooperative Information Agents: Best Papers of CIA 2001.
23. Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter – Designing Conventions for Automated Negotiation Among Computers*. MIT Press, 1994.
24. Michael Schumacher. *Objective Coordination in Multi-Agent System Engineering – Design and Implementation*, volume 2039 of *LNAI*. Springer-Verlag, April 2001.
25. Reid G. Smith. The Contract Net Protocol: High level communication and control in distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980.
26. William L. Thomson. Cooperative models of bargaining. In R. Aumann and S. Hart, editors, *Handbook of Game Theory*, pages 1238–1284. North-Holland, 1994.
27. Robert Tolksdorf. Models of coordination. In Andrea Omicini, Franco Zambonelli, and Robert Tolksdorf, editors, *Engineering Societies in an Agent World*. Springer-Verlag, 2000.
28. Mirko Viroli and Andrea Omicini. Coordination as a service: Ontological and formal foundation. In Antonio Brogi and Jean-Marie Jacquet, editors, *Foundations of Coordination Languages and Software Architectures – Papers from FOCLASA'02*, volume 68 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B. V., 2002.
29. Frank von Martial. *Co-ordinating Plans of Autonomous Agents*, volume 661 of *Lecture Notes on Artificial Intelligence*. Springer-Verlag, Berlin, 1992.
30. Peter Wegner. Why interaction is more powerful than computing. *Communications of the ACM*, 40(5):80–91, May 1997.
31. Workflow Management Coalition home page. `http://www.wfmc.org/`.
32. Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
33. Michael J. Wooldridge, Gerhard Weiss, and Paolo Ciancarini, editors. *Agent-Oriented Software Engineering II*, volume 2222 of *Lecture Notes on Computer Science*. Springer-Verlag, January 2002. 2nd International Workshop (AOSE 2001), Montreal, Canada, 29 May 2001, Revised Papers and Invited Contributions.
34. Franco Zambonelli, Nicholas R. Jennings, Andrea Omicini, and Michael J. Wooldridge. Agent-oriented software engineering for Internet applications. In Omicini et al. [17], chapter 13, pages 326–346.