https://doi.org/10.1142/9789811239922_0003

Chapter 3

Social Commitments for Engineering Interaction in Distributed Systems

Matteo Baldoni, Cristina Baroglio, Roberto Micalizio and Stefano Tedeschi
Università degli Studi di Torino Dipartimento di Informatica

3.1 Introduction

Multiagent Systems (MAS) [1] are an effective choice for the design and development of distributed systems that involve components which act independently (i.e., autonomously). The scenario, in fact, calls for the use of four major software engineering techniques to cope with size and complexity — namely, modularity, distribution, abstraction, and intelligence (i.e., flexibility) — and the MAS paradigm encompasses all of them [2].

Nowadays, agent-oriented software engineers can choose from a substantial number of agent platforms (see, e.g., [3] for an overview). Tools like JADE [4], TuCSoN [5], DESIRE [6], and JaCaMo [7] all provide coordination mechanisms and communication infrastructures. However, the limit of the best-established platforms is a lack of abstractions for explicitly modeling interaction as a first-class entity. All of them provide communication infrastructures, mainly as message passing, but none of them encompass a mechanism to explicitly represent and manipulate the relationships created by the agents during their interaction. The lack of such a mechanism impairs the agents to reason on how to get their goals by engaging with others. As noted in [8], MAS, by their nature, do not include a centralized control mechanism, and agents are expected to reason about what interactions to engage with others. This peculiarity endows agents with a significant flexibility of action, but the current platforms, instead of explicitly accounting for a "society" layer [9], for regulating and norming how agents can act and interact within the system, their constraints and the system laws, force the

projection of the regulations directly inside the behaviors of the agents. As a consequence, MAS infrastructures do not really preserve the agents' autonomy and they do not fit the high degree of decoupling which is expected of the agents. This choice overly ties agent implementation with a negative impact on software reuse, and also on the realization of open MAS, that is, MAS that agents, possibly heterogeneous and developed by independent parties, can dynamically join/leave. Thus, when interaction is hard-coded as an exchange of messages, the advantage of having autonomous agents is significantly reduced. Instead, if social relationships are represented as first-class entities, that is, resources that agents can manipulate, they also become synchronization tools, reducing coupling between agents and improving flexibility (for a detailed discussion see [10]).

As a first contribution of this chapter, we illustrate the advantages of including an explicit representation of the *social relationships* that tie the agents, and in particular the positive impact such a new explicit representation would have on code modularity and interaction flexibility. We practically demonstrate these advantages when social relationships are modeled as *social commitments* [11] and *reified* as resources in the environment. A social commitment is a promise that an agent (debtor) makes to another one (creditor) to bring about some condition of interest. It is, therefore, a relationship with a distributed nature, directed from the debtor to the creditor. It engenders rights on both sides. For example, the creditor has the right to complain if the commitment is not fulfilled and the debtor has the right to expect its performance of the action concerned to be accepted as the fulfilment of this commitment. So, social commitments have a normative power, yielding obligations and expectations: by withdrawing from a social commitment, an agent violates an obligation, and frustrates expectations and rights [8, 12].

In a system where interaction is ruled by a message-passing protocol, the parties are designed so as to be compliant to the protocol. The expectations that one party has on the others, as well as the corresponding obligations, are granted "extra-program", in the sense that they are yielded by the protocol (i.e., the standard) to which each of them *declares* to comply to. Social commitments capture such a feature by their own nature (indeed, the above declaration is, by itself, the commitment of the party to adhere to the protocol specification). Building upon this and following the direction postulated in [13], in this work we use them as building blocks for creating the interaction standards themselves — possibly in a dynamic way, depending on the agent goals and on the context. Social commitments, created by their debtors, are promises that such agents make to behave in a way that complies to the achievement of the expressed condition, resulting into an obligation for the involved agents. All the agents that can observe the commitment will now expect something specific to occur sooner or later. The advantage of

social commitments, w.r.t. message-passing protocols, is that they are created by the agents depending on their own objectives and decisions, this promotes some important software engineering properties as we discuss below.

As a second contribution, we show that when commitments are realized as resources, that can be manipulated directly by the agents, the system as whole can gains some important benefits. In particular, agents can use commitments to reason upon their interactions, and can determine how and when engaging with other agents to achieve goals of their own interest. This is a significant contribution for the MAS research area, where, despite one of the most important characteristics of the MAS paradigm being agent situatedness, most studies are focussed only on features of the agents. At the same time, those that put forward the need of representing the environment typically do not provide a representation of the process, by which data evolve, in a form that can be reasoned about. This hampers interaction because agents are unable to create expectations on how others will act upon data that are part of the environment. Proposals like [14, 15] introduce first-class abstractions for the environment, to be captured alongside agents themselves. In particular, [15] states that "the environment is a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources." This proposal brought to the Agents & Artifacts (A&A) meta-model [16], and its implementation CArtAgO [17]. A&A enables the agents to mediate their interactions by means of shared computational resources — the artifacts — which can encapsulate information. Hence, they have the advantage of a clear separation between the agent deliberative cycles and the data upon which this cycle is carried out. However, artifacts do not encompass a normative dimension, and this prevents agents to create expectations about others.

A similar proposal has been put forward by [18] in the business process setting. They propose business artifacts as business-relevant objects that are created and evolve as they pass through business operations. Business artifacts include an information model of the data, and a lifecycle model. The latter captures the key states through which data evolve and their transitions, and it is used both at runtime (to track the evolution of business artifacts), and at design time (to distribute tasks among the processes that operate on a business artifact). Also business artifacts, however, do not provide any link to a corresponding normative understanding, thus making impossible for the agents (processes) to leverage this knowledge for reasoning about how to act.

Last but not the least, social commitments, and their normative dimension, allow one to define the agent programming patterns. The commitment lifecycle, in fact, provides a clear direction on how agents participating to an interaction

should be implemented. We present four programming patterns that relate the individual agent's goals with her role, as debtor or creditor, in a set of commitments. These patterns help a programmer implement agents' behaviors in the following cases: create a new commitment to foster cooperation from others (*entice*), take part into an interaction by moving a commitment towards its satisfaction (*cooperate*), withdraw an offer when it is no longer useful (*withdraw*), and release a commitment when the made offer is not appealing (*give up*). Notably, these patterns are independent of any specific agent platform because they only depend on the standardized lifecycle of commitments. We exemplify the use of the patterns for agent programming in the context of a realistic hiring scenario using the JaCaMo+ platform [19].

The chapter is organized as follows. In Sec. 3.2 we exploit a motivating example for positioning our contribution against the current approaches for dealing with goal distribution and coordination of a number of independent execution threads. Section 3.3 provides some background notions about commitments that are essential for the proposal we present in Sec. 3.4, and highlights its advantages from a software engineering perspective. Agent programming patterns driven by the commitment lifecycle are presented in Sec. 3.5, followed by an example about the use of these patterns in JaCaMo+ in Sec. 3.6.

3.2 Background on Goal Distribution and Coordination

Not always results can be achieved in isolation, but require an interaction with others. This happens, for instance, when an agent must rely on an action performed by another agent, for necessity or for convenience. However the two agents could likely have different goals, which then have to be accommodated during the interaction. The possibility to reach each actor's goal depends on the evolution of the interaction and, thus, an effective coordination is crucial [2].

Example 3.1 (Item Purchase). Let us consider, as an illustrative example, a purchase scenario, involving a customer willing to buy a given quantity of an item from a merchant. The scenario is inspired from the well-known Netbill protocol [20]. When the customer requests a quote for the items from the merchant, the latter must send the quote. If the customer accepts the quote, the merchant sends the items, then waits for an electronic payment. We assume that the items cannot be used without a decryption key, sent by the merchant after the payment together with a receipt.

Coordination in Business Processes. A possible way to model Example 3.1 is by representing the customer and the merchant as two interacting BPMN (Business Process Model and Notation) Processes [21], each one pursuing a different objective, as reported in Fig. 3.1. The progression of each process depends on the other one, and requires coordination. In particular, the ultimate goal of the merchant (i.e., to complete the purchase) depends on the reception of several messages from the customer. However, the two processes are "separate", even if interacting (e.g. they might have been developed within different companies). For this reason no assumption can be made on the actual behavior of the customer. This is the reason why the customer's process is not shown in Fig. 3.1.

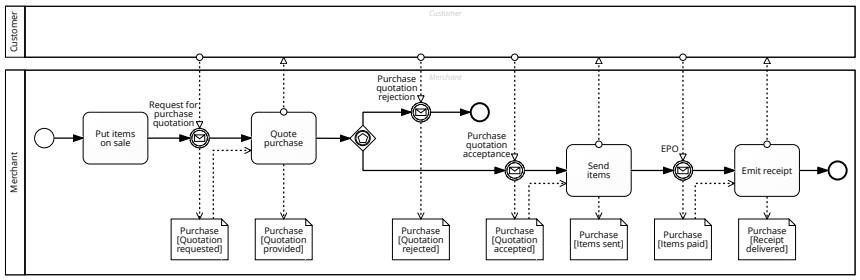

Fig. 3.1: BPMN diagram of the *Purchase* scenario.

A substantial limitation of this approach is that it does not capture well how the information relevant for the progression of the purchase evolves. Such information can be modeled as a *data object*, whose state evolves along with the execution of the process, in order to support the *data consistency*, i.e. that the data used in each step of the process does not contain contradictions w.r.t. the data used and produced in the previous steps. However, this is not sufficient because neither the process specification nor the data object specification include "causal" relationships between the actions of the various agents involved [22]. For instance, an agent may know that by paying an item the purchase will change state to "paid", but this is not enough to allow the agent to have expectations about the item being shipped as a consequence of paying. So, why paying? It is not a goal of the agent to make the purchase pass to the state "paid", but rather to have the item delivered. Paying is instrumental to delivery but delivery depends on other agents. What we need here is something that allows the buyer agent to legitimately have the expectation that the item will be delivered after paying. In business processes, the interaction between the two processes is loosely modeled in terms of message exchanges, thus not allowing each one to create expectations about other's behavior. When a business goal is split over a set of interacting processes, each of these

will realize only a part, and, in order to realize such a part, it will generally depend on the achievement of sub-goals that are realized by other processes. The synchronization realized in BPMN through message-passing or by the introduction of shared data storages does not capture coordination in high-level terms, that enable the agents, that dynamically join the system, to understand the obligations they can legitimately draw as consequences of their actions.

A more *data-oriented* approach is to model the purchase as a CMMN (Case Management Model and Notation) Case [23], as illustrated in Fig. 3.2. CMMN is a graphical notation used for capturing the handling of particular situations (i.e. *cases*) requiring various activities that may be performed in an unpredictable order in response to evolving situations. CMMN overcomes some of the limitations of BPMN, allowing to model work efforts which are less structured and depend on events, possibly occurring as a result of the evolution of data. A CMMN case is represented as a set of activities, possibly depending one on another, to be preformed in response to certain events. The activity execution makes the case evolve, running across multiple *milestones*. Despite allowing to encompass the evolution of data in the modeling of processes, CMMN still fails in explicitly representing the mutual engagements between the actors involved in a distributed processes.

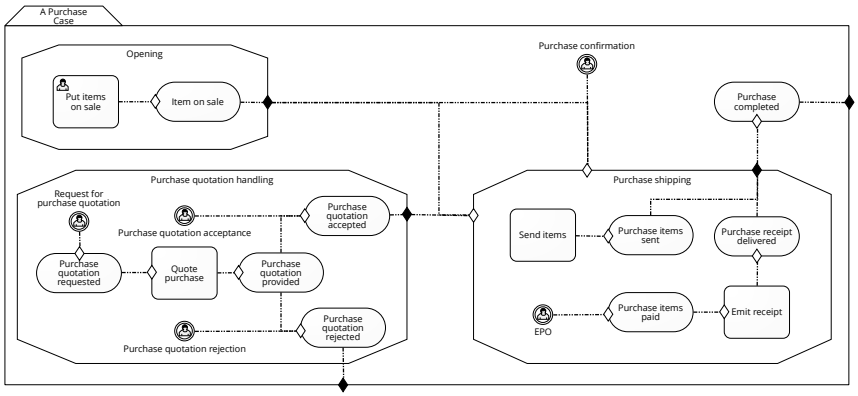

Fig. 3.2: CMMN diagram of the *Purchase* scenario.

Coordination with Business Artifacts. An alternative approach to cope with coordination is the adoption of *business artifacts*. Business artifacts add an information layer concerning both the structure and the lifecycle of the data they encompass. Some authors [24] propose to use them as a means to combine process

engineering with data engineering. Still, as explained in [25], they do not support coordination satisfactorily. To understand why, let us focus on the BALSA (Business Artifacts with Lifecycle, Services, and Associations) methodology [26], that we consider as a significant representative of the business artifacts approaches. Here, coordination among of business processes is tackled by relying on *choreographies*, regulating different business processes, which access to a same business artifact. Choreographies realize a form of *subjective coordination* [27]. This means that each business process needs to include also the "interaction logic" of the choreography role it plays along with its business logic. In other words, there is a strict coupling among the implementations of all the interacting processes. Consequently, the design and implementation become more complex, and the possibility of reusing the same process in different contexts is reduced.

Coordination in Multiagent Systems. Since the early proposals for MAS programming, organizations have been seen as metaphors for modularizing the code. Organizations, in fact, provide an overall abstraction of the task the agents have to achieve. In Gaia [28, 29], for instance, organizations are characterized by two features: a set of roles and a set of interactions among roles. Here interactions are seen as protocol definitions; where a protocol is "an institutionalized pattern", namely, a pattern that has been formally defined [29]. The pattern, thus, defines the rules (i.e., *norms*) through which an institutional reality, i.e. an interaction scope delimiting these norms, takes shape and evolves [30]. This institutional reality is the actual means of coordination of the agents: its constituent elements, the *institutional facts*, have a social meaning, by way of norms, that is shared and understood by all the agents participating to the interaction. Agents, thus, act so as to bring about those institutional facts that represent their goals or duties towards others. In other terms, when norms are explicitly represented and known by all the participants, it is possible to create *expectations* about the behavior of others in response to given messages, and this allows determining when and how to act. Indeed, coordination is all about expectations: an activity can fruitfully be carried out by many parties when there is a clear understanding on what each one should do and when. So one party will wait for the completion of the task by another party before starting its part. On the same grounds, the party who is first to act is confident that another party will continue from where it stopped.

Tools. The development of MAS relies upon a number of frameworks. JaCaMo [7] is a programming platform that integrates agents, environments and organizations. It is built on top of Jason [31], an agent programming language,

CArtAgO [32] for programming environments, and $\mathcal{M}$OISE [33] for programming organizations. 2COMM [10, 34] is a middleware that provides functionalities to reify social expectations as CArtAgO [32] artifacts, that are made available to the agents in the environment where they are situated. Currently, 2COMM supports social relationship-based agent programming for JADE [4] and JaCaMo [7] agents through two dedicated connectors. 2COMM provides the classes for representing commitments, maintained into artifacts, as well as roles. Moreover, the infrastructure automatically handles the commitment progressions according to the events occurring in the environment. In particular, events that are relevant for the progression of commitments are encoded as *social facts*. 2COMM maintains a trace of all the social facts asserted in a given interaction and updates the involved commitments accordingly.

3.3 Background on Social Commitments

We propose to model interaction explicitly by means of *social commitments* [22, 35], intended as first-class objects that can be used for agent programming. A social commitment $\mathsf{C}(x, y, s, u)$ models the directed relation between two agents: a *debtor* x and a *creditor* y. The debtor commits to its creditor to bring about the consequent condition u when the antecedent condition s holds. Both conditions are conjunctions or disjunctions of events and commitments, and concern the *observable* behavior of the agents, as advocated in [36] for social relationships among autonomous parties.

In this chapter we will assume that antecedent and consequent conditions are expressed in *precedence logic* [37, 38], an event-based linear temporal logic for modeling web services' composition. It deals with occurrences of events along runs (i.e., sequence of instanced events). Event occurrences are assumed to be non-repeating and persistent: once an event has occurred, it has occurred for the whole execution. The logic has three operators: "$\vee$" (choice), "$\wedge$" (co-occurrence), and "$\cdot$" (before). The *before* operator allows constraining the order with which two events must occur; e.g., $a \cdot b$ means that a must occur before b, but the two events do not need to occur one immediately after the other.

Commitments are proactively created by debtors, who, by doing so, manifest both the agent's engagement to bringing about the consequent condition, and the agent's awareness of the condition (antecedent condition) after which it will be held to bring about the consequent. Generally, the antecedent condition will be a condition of interest for the debtor. When the creditor has some degree of control on such a condition, the commitment becomes a tool supporting the interaction. For instance, let us consider the commitment $\mathsf{C}(customer,$

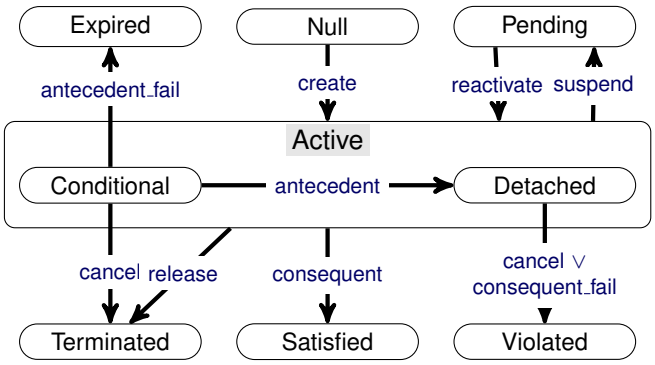

Fig. 3.3: Commitment life cycle [39].

$merchant$, purchaseItemsShipped, purchaseItemsPaid). Here, $merchant$, who is interested in being paid, can leverage the commitment by $customer$ to do so after the shipment: by making purchaseItemsShipped become true, the commitment is detached and, then, $customer$ is held to pay, thus making purchaseItemsPaid become true.

Commitments have a standardized lifecycle, formalized in [39] and depicted in Fig. 3.3. As soon as a commitment is created, it is *active*. Active commitments can further be in two sub-states: *conditional* when neither the antecedent, nor the consequent have occurred, and *detached* as soon as the antecedent becomes true. A commitment is *violated* either when its antecedent is true, but its consequent becomes false or when it is canceled by the debtor when detached. Conversely, it is *satisfied* when the consequent occurs and the engagement is accomplished. It is *expired* when it is no longer in effect; i.e., if the antecedent condition becomes false. Finally, a commitment becomes *terminated* when the creditor releases the debtor from it or when it is canceled, being still conditional.

Social commitments can be manipulated by the agents through some standard operations: namely, *create*, *cancel*, *release*, *discharge*, *assign*, and *delegate* [35]. *Create* instantiates a commitment, changing its status from *null* to *active*. Only the debtor of a commitment can create it. Conversely, *cancel* revokes the commitment, setting its status to *terminated* or *violated*, depending on whether the previous status was *conditional* or *detached*, respectively. Again, only the debtor can cancel a commitment. The rationale is that a commitment debtor is allowed to change its mind about the accomplishment of the consequent only until the antecedent has not occurred, yet. *Release*, in turn, allows the creditor (and only the creditor) to terminate an active commitment. Release does not mean success or

failure of the given commitment, but simply eliminates the expectation put on the debtor. *Discharge* satisfies the commitment. In accordance with [35], we assume that this operation is performed concurrently with the actions that lead to the consequent occurrence. *Delegate* changes the debtor of a given commitment and can be performed only by the new debtor. *Assign*, finally, transfers the commitment to another creditor and can be performed only by the actual one. An active commitment can be further suspended, i.e., put in the *pending* status, by means of the *suspend* operation. Conversely, *reactivate* sets a pending commitment as active, again.

Commitments have a *normative value* because the debtor of a detached commitment is expected to bring about, sooner or later, the consequent condition of that commitment otherwise it will be liable for a violation. The normative value of commitments, i.e. the fact that debtors should satisfy them, creates social expectations on the agents' behavior. A commitment is taken by a debtor towards a creditor on its own initiative. Agents can decide whether satisfying the obligation entailed by the commitment, once detached. An agent creates commitments towards other agents while it is trying to achieve its goals (or precisely with the aim of achieving its goals) [40]. The creation of a commitment starts an interaction of the debtor with its creditor that coordinates, to some extent, the activities of the two, thus supporting the achievement of goals that an agent alone could not achieve. Considering *interaction*, the difference between obligations and commitments, as norms, is that an obligation is a system level norm while a commitment is an agent level norm. At system level, something happens and an obligation is created on some agent. At the agent level, an agent creates a conditional social commitment towards some other agent, based on its own beliefs and goals [40]. The creditor agent will detach the conditional commitment if and when it deems it useful to its own purposes, thus activating the obligation of the debtor agent. So, conditional commitments play a fundamental role in the realization of interactivity, intended as the fact that *a message relates to previous messages and to the way previous messages related to those preceding them* [41]. In other words "there is a causal path from the establishment of a commitment to prior communications by the debtor of that commitment. Obligations by contrast can be designed in or inserted by fiat" [22, Sec. 4.4].

Notably, social commitments are often used for attributing semantics to interaction protocols (e.g. [42–47]). Differently from these works, we use commitments to realize a *relational representation of interaction*, where agents, by their own action, directly create normative binds (represented by social commitments) with one another, and use them to coordinate their activities.

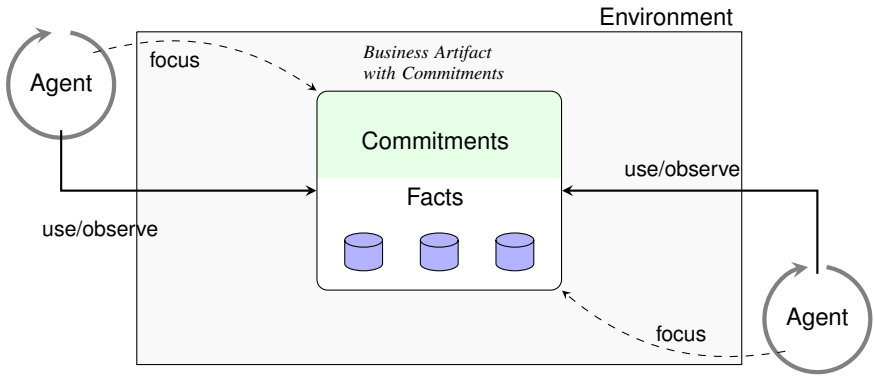

Fig. 3.4: Conceptual view of commitment mediated interaction.

3.4 Business Artifacts with Social Commitments

Figure 3.4 shows a conceptual schema of the proposed solution. Agents interact by using and observing shared artifacts that, besides data (i.e. facts), also encapsulate the commitments created by the agent themselves. More precisely, when an agent needs to interact with others, it is asked to focus on an appropriate artifact available in the environment. Possibly, an agent can be focused on more artifacts at the same time. As a consequence of the focus operation, the agent has access to the state of the artifact, that is, all the information involving the existing commitments and facts that have been asserted, so far, as a consequence of the operation performed upon the artifact itself. Indeed, the focus gives also the agent the capability of acting upon the artifact by creating new commitments or by making other existing commitments progress by asserting facts that corresponds to their antecedent/consequent conditions. By focusing on an artifact, the agent will be notified of any change occurring in the artifact state, such as the assertion of new facts, and the progression of the state of some commitments.

To make this discussion more concrete, let us consider again our simple purchase scenario. We show that when the interaction between the merchant and the customer is expressed in terms commitments, the agents have the means for reasoning upon their engagements, and hence can deliberate the proper operations for making the interaction progress towards the result they are interested to.

First of all, the environment encompasses a *Purchase* artifact, see Fig. 3.5. Both the merchant and the customer need to include such artifact in their scope for having access to it and also for being notified of the changes occurred to it (*focus*). Each agent has its own goal. The merchant aims at gaining some money,

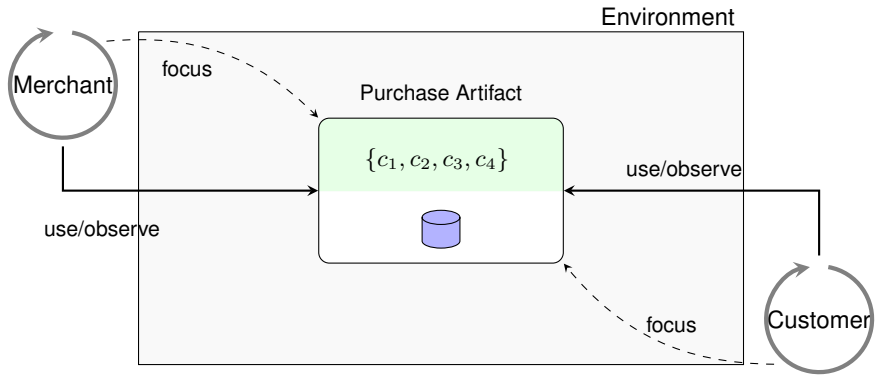

Fig. 3.5: The conceptual view of the purchase scenario.

whereas the customer aims at having some needed items. The *Purchase* artifact represents the medium through which the interaction between the two agents can take form. The expectations about the interaction are shaped as an evolving set of commitments. So, for instance, by committing to pay for the items, should the merchant deliver them, the customer makes clear which expectations the merchant can legitimately draw on its counterpart of the interaction. The created commitment is stored within the *Purchase* artifact. This is exactly the kind of information that is missing in approaches based on (business) artifacts where the normative dimension is not explicitly represented. Agents just react to signals or state changes, but cannot figure out a global picture of the interaction and take advantage of this picture for deciding how to intervene. By relying on commitments as manipulable resources, instead, the agents have the means for creating expectations on how the others will behave as a consequence of their actions, and hence can decide the best course of action that brings them to their goal.

The normative power of commitments emerges not only by the possibility of creating expectations, but also by detecting when these expectations are not met, causing thus a commitment violation. In fact, when the customer commits to pay for the goods, the merchant gains the right to claim against the customer in case the payment is not performed. In other terms, the actions the merchant does upon the *Purchase* artifact may put obligations on the customer, who is pushed to act so as to satisfy her commitments lest incur in violations which may lead to sanctions.

The whole purchase can be effectively modeled by the following set of commitments:

- c_1 : C($merchant, customer, c_2$, purchaseItemsShipped)
- c_2 : C($customer, merchant$, purchaseItemsShipped, purchaseItemsPaid)

- c_3 : C($merchant, customer$, purchaseQuotationRequested, purchaseQuotationProvided)
- c_4 : C($merchant, customer$, purchaseItemsPaid, purchaseReceiptDelivered)

Commitments c_1, c_3 and c_4 are to be created by the merchant at the beginning of the interaction. In particular, c_1 is a *nested* commitment, that is, a commitment that includes other commitments in its conditions. Specifically, c_1 expresses the promise, by the merchant, of delivering the items in case the customer promises to pay for them (commitment c_2). When the customer actually creates c_2, on the one side c_1 is detached, and consequently the merchant is driven to ship the items; on the other hand, the customer promises to the merchant to pay when these items will be delivered. Commitment c_3 is used to model the negotiation phase, it expresses that, should the customer request a quotation for the purchase, the merchant will answer with the quotation. Finally, c_4 is the commitment from the merchant to the customer to send the receipt upon the payment. Note that the events mentioned in the antecedent and consequent of the commitments, are indeed facts (i.e., data) asserted within the artifact state as a consequence of predefined operations made available to the agents by the artifact itself. Therefore, by acting upon the *Purchase* artifact, both merchant and customer have the chance not only to create commitments, but also the make them progress along their lifecyle. In the example, the commitments completely shape the expected evolution of the interaction between the involved agents by making explicit the behavior they are expected to stick to. As such, commitments provide a standard to define patterns of interaction mediated by the environment in which agents are situated.

Objective Coordination and Decoupling. Interaction is surely one of the most critical dimension to be addressed in distributed systems. To cope with the problem, message passing protocols are widely used in many multiagent platforms (see e.g., JaCaMo and JADE), and even in BPMN. A disadvantage of message protocols, however, is that the interaction logic is intermingled with the agent control logic. That is, the coordination is modeled in a *subjective* way. This lack of concern separation hampers software modularity and reuse. For promoting software modularity, coordination should be implemented *on the side of the callee* [48]. Namely, the coordination should be implemented outside the agents, in a class/resource that is accessed concurrently by the agents being coordinated.

Our proposal, on the contrary, enables a form of *objective coordination* [27, 49], where coordination is addressed outside the interacting agents. Objective coordination enables a clear separation between the implementations of the business logic and of the coordination logic by explicitly representing the environment where the agents operate. In our proposal, we meet this property by reifying

commitments, that possess a normative power, inside a dedicated coordination artifact, which is external to the agents. The interaction logic is thus encoded into a single component (i.e., the coordination artifact), and is not distributed and intermingled within the agents' code. A positive consequence is that the implementation of environment resources (i.e., artifacts) and of the agents' processes can be carried out and verified in *isolation*. That is, not only agents are strongly decoupled with each other, as their interaction is mediated by an artifact, but there is also a form of decoupling between agents and artifacts. The interaction logic encoded within an artifact can in fact be changed without the need to change the agents as far as the artifact interface (i.e., the set operations), remains unchanged. For instance, in the purchase scenario, it may be possible to change the interaction logic by adding a timeout on the detachment of commitment c_1. The rationale would be to release the merchant if the customer does not accept within a finite time interval (which is considered an anomalous situation). This, however, is a characterization of the *Purchase* artifact that has no a direct impact on the participants.

Decoupling enables also the development and verification of agents and artifacts in isolation (see below).

Flexibility. Some approaches rely on automata to model the stages through which an interaction progresses. For instance, the *interaction component* [50, 51], for the JaCaMo platform, enables both agent-to-agent and agent-to-environment interaction, providing guidelines of how a given organizational goal should be achieved, with a mapping from organizational roles to interaction roles. Guidelines are encoded in an *automaton-like* shape, where states represent protocol steps, and transitions between states are associated with (undirected) obligations: the execution of such steps creates *obligations* on some agents in the system, which can concern actions performed by the agents in the environment, messages that an agent sends to another agent, and events that an agent can perceive (i.e., events emitted from objects in the environment). Business artifacts [18] are another example where the lifecycle of data are represented as automata. By acting upon these artifacts, processes cause automata state changes. The specification of interaction via automata, however, shows a rigidity that prevents agents from taking advantage of the opportunities and of handling exceptions in dynamic and uncertain multiagent environments [52, 53]. Agents are, in fact, confined to the execution sequences provided by the automaton.

On the contrary, commitments do not impose any strict ordering. As far as any detached commitment progresses to satisfaction, the interaction can be considered successful. It must be noticed that a commitment could even be satisfied

before its detachment, this opens the way to deal with "cases" more flexibly and in a way that is paralleling CMMN. Let us suppose, for instance, that we want to capture an emergency situation where some medical equipment is needed urgently. The customer may create commitment c_2 even before knowing the asked quote. That is, there is not a strict ordering between first asking for a quote and then accepting it, as it may happen in an automaton-based model. More interesting, the merchant, knowing the emergency conditions, and trusting the customer, may ship the equipment even before the customer creates commitment c_2. This means for the merchant to bring about the consequent condition of a commitment that has not even been detached. When eventually the customer creates c_2, such a commitment is already detached, and hence the customer pays for the equipment.

Commitments, thus, when used as directly manipulable resources, enable the agents to decide how and when be engaged in an interaction, in ways that are substantially more flexible than approaches based on message passing and automata.

Comparison with Existing Frameworks. We now briefly compare the proposed approach with two of the most established frameworks for multi-agent systems, namely JADE and JaCaMo, already introduced in Sec. 3.2.

A first positive consequence of an explicit, directly manipulable, representation of social commitments is that agents can be programmed by following a uniform schema that is agent-platform independent. This is not, however, the unique advantage. Table 3.1 synthesizes a comparison between JADE and JaCaMo, highlighting aspects that are improved or added by an explicit representation of commitments.

Table 3.1: Comparison among JADE, JaCaMo, and improvements provided by commitments.

	JADE	**JaCaMo**	**Social Commitments**
Programmable interaction means	X	✓	✓
Notification of social relationships of interaction	X	X	✓
Interaction/agent logic decoupling	X	X	✓
Reasoning on expected behaviors	X	X	✓
Definition of structured schemas for interaction	✓	X	✓
Runtime interaction monitoring	X	✓	✓
Agent programming aware of social relationships	X	X	✓

- *Programmable interaction means*: The reification of social commitments as artifacts embodied in the environment significantly improves JADE, where programmable interaction means are missing. On the other hand,

JaCaMo already integrates the CArtAgO component and, hence, already includes programmable artifacts used as means of interaction.

- *Notification of social relationships of interaction*: Our proposed business artifacts implement the commitment lifecycle. Any change of state in the artifact, thus, corresponds to a change in the interaction state. For this reason, a form of automatic notification to the agents of such changes (concerning social relationships) is enabled. This is not provided by JADE nor by JaCaMo.
- *Interaction/agent logic decoupling*: In JADE and JaCaMo, no specific abstractions are used for modeling and implementing the interaction logic. In JADE, agents interact by exchanging messages. In JaCaMo, agents can modify an artifact for communicating with others. In both cases, however, the semantics of the message or of the artifact operation must be encoded within the interacting agents. With dedicated artifacts encapsulating commitments, when an agent acts upon the environment, the meaning, and implications, of such an action is modeled via a state change occurring in a specific commitment. It follows that the logic of the interaction is not spread inside the agent code, but within the artifacts.
- *Reasoning on expected behaviors*: As we have said, commitments have a normative power. Neither JADE nor JaCaMo can rely upon an analogous mechanism.
- *Definition of structured schemas for interaction*: The commitment lifecycle allows relying on a general schema for programming agents that is independent of the agent-platform.
- *Runtime interaction monitoring*: Although we will not discuss this aspect in detail, the use of dedicated artifacts facilitates the implementation of monitors that supervise the whole interaction occurring in the system. It would be sufficient, in fact, to have an agent monitoring each artifact of and hence capturing every change of state in the commitments. Of course, this is a direct consequence of having social relationships explicitly modeled as first-class elements and not implicitly modeled via message passing as in JADE. In JaCaMo, a monitor could also be implemented as an observer of the artifacts in the environment.
- *Agent programming aware of social relationships*: The proposed approach guides a programmer in developing an agent that is aware and compliant with its engagements, as we will show in detail in the following section.

3.5 Patterns for Programming Agents with Social Relationships

Commitments bring along advantages also from the point of view of agent software development. In fact, their standardized lifecycle can be used as a reference for developing agents that use them. Intuitively, an agent involved in a commitment, either as debtor or as creditor, should act so as to make the commitment state progress toward satisfaction w.r.t. its lifecycle (consequent condition achievement). Thus, the lifecycle can provide a hint to the programmer on what agent behaviors she needs to implement. We can think of the commitment lifecycle as a sort of bank of programming patterns, that can be defined taking also into account the goals the two commitment participants aim at. The relation between goals and commitments has been deeply studied in [40, 54], where the authors propose several rules of practical reasoning that agents can use to determine, e.g., when it is convenient to create a commitment, or detach an existing one, when they should act to satisfy a commitment, and so on. Intuitively, these practical rules help agents to align their intentions by means of commitments, so as that all the agents participating to an interaction can achieve their desired goal by exploiting the cooperation of the others.

We take advantage of these practical rules, and propose some agent programming patterns where, by means of the direct use of commitments as interaction resources, an agent can induce the cooperation from others

The aim of the following patterns is, thus, to implement one (or more) social behaviors that allows an agent ag to achieve a goal G by cooperating with other agents. In the following, we assume that an agent has already focused on an artifact of interest, and use the term *social state* for denoting the state of such an artifact, that is, the collection of commitments and facts (data) maintained within the artifact.

(1) *Entice*:

- *Intent*: Finding the cooperation of another agent by making an offer.
- *Applicability*: When, by inspecting the social state, the agent does not discover cooperation offers by other agents that support the achievement of its goal adequately, then the agent creates an offer itself.
- *Implementation*: Implement a behavior that creates a commitment $\mathsf{C}(ag, other, G, q)$, where ag is the help-seeking agent, $other$ is another agent focused on the artifact, and q is a condition that ag can bring about, either directly or via the cooperation with others.

(2) *Cooperate*:

- *Intent*: taking part to a collaboration by making a commitment progress towards satisfaction.
- *Applicability*: when the agent is either the debtor or the creditor of an active commitment.
- *Implementation*:
  - When ag is the creditor: implement a behavior that is triggered by the creation of a commitment $c : \mathsf{C}(other, ag, p, G)$ and that brings about p, so as to detach c.
  - When ag is the debtor: implement a behavior that is triggered by the detachment of a commitment $c : \mathsf{C}(ag, other, G, q)$ and that brings about q so as to satisfy c.

(3) *Withdraw offer*

- *Intent*: Retire an offer of cooperation that is no longer needed. Agent ag has tried to entice the cooperation of some other agent by creating $\mathsf{C}(ag, other, G, q)$, but now it realizes that G is no longer needed. Retiring the offer, ag avoids to consume resources for achieving q in exchange of the undesired G.
- *Applicability*: When goal G is no longer necessary (local decision making of ag), and $\mathsf{C}(ag, other, G, q)$ is still conditional in the social state.
- *Implementation*: Implement a behavior that cancels the conditional commitment $\mathsf{C}(ag, other, G, q)$ currently stored in the social state.

(4) *Give up*

- *Intent*: Release the obligation upon the achievement of a goal, when such a goal is no longer desired. Agent ag has taken part to cooperation by detaching commitment $c : \mathsf{C}(other, ag, p, G)$. The original intent of ag was to get G, and by performing p it has caused the detachment of c. Now the obligation of bringing about G is pending on $other$. While c is detached, however, ag may realize that G is no longer needed. It may even been deleterious, and so can decide to release the commitment, that is, can remove the obligation on $other$ to bring about G. More generally, a creditor can release a commitment even when the commitment is still conditional: ag is not interested in the offer made by the debtor and brings the commitment to a terminal state.
- *Applicability*: When goal G is no longer necessary, and the active (i.e., either conditional or detached) commitment $c : \mathsf{C}(other, ag, p, G)$ is stored in the social state.
- *Implementation*: Implement a behavior that releases commitment c.

It's worth noting that the applicability of the *Withdraw offer* and *Give up* patterns is the same. The two patterns represents different ways in which debtor and creditor can leverage an existing commitment. Being the commitment still conditional, should the debtor be not interested anymore in the antecedent, it can cancel it by means of *Withdraw Offer*. Conversely, should the creditor be not interested in the consequent, it can release the debtor from its commitment by means of *Give up*.

The above patterns relate agent goals with the engagements the agent is involved in. In particular, *Entice*, *Withdraw offer* and *Give up*, are patterns "from goal to commitment", that is, on the necessity of achieving or dropping a goal, an agent acts directly on a commitment by means the operations create, cancel, and release, respectively. Pattern *Cooperate* encompasses both a stance "from goal to commitment" when an agent is creditor and acts upon a commitment to bring about the antecedent condition, so as to detach it. And it has also a "commitment to goal" stance when the agent is a debtor and react to the detachment of a commitment by adding the consequent condition among the goals it has to pursue. Of course, the patterns we have introduced cover just a subset of the possible state transitions in a commitment lifecycles. Other patterns leading to the violation of a commitment or to its suspension are possible, but for the purpose of the present paper we have focused on those patterns that promote coordination and lead to the satisfaction of a commitment.

Let us explain how these patterns are applied to the previously introduced *Purchase* scenario. For the sake of simplicity, we use an abstract language based on ECA rules (Event, Condition, Action), for expressing the agents' behaviors. In Sec. 3.6 we will show an actual implementation in JaCaMo+ [19] (i.e. JaCaMo extended with 2COMM [10, 34]). Here, we adopt the following syntax for ECA rules: $ON\ event\ IF\ condition\ THEN\ action$. The event denotes the trigger for activating such a rule, in general it is a goal that the agent wants to achieve or an event that must be properly treated by the agent. The condition is a contextual circumstance that must hold for the rule to be actually fired. Finally, the action is a course of operations that modify the environment so as to obtain a desired effect.

Let us take the $merchant$'s perspective, whose ultimate goal is to conclude the purchase and get paid. It can do so by exploiting the *Entice* schema, in such a way to create commitments c_1, c_3 and c_4, defined in Sec. 3.4, to make an offer to the potential customer. This behavior could be captured by the following ECA rule:

ON needMoney IF haveItemsToSell
DO create(c_1 : C($merchant$, $customer$, c_2, purchaseItemsShipped));
create(c_3 : C($merchant$, $customer$, purchaseQuotationRequested, purchaseQuotationProvided));
create(c_4 : C($merchant$, $customer$, purchaseItemsPaid, purchaseReceiptDelivered)).

The use of the *Entice* pattern is strictly connected to the use of the *Cooperate* one (as debtor), to tackle the relevant commitment state changes (i.e. their detachment). For c_1, the corresponding ECA rule would be:

ON detached(c_1) DO $...ship...$ assert(purchaseItemsShipped).

The triggering event is the detachment of commitment c_1, that is, $customer$ has confirmed the purchase by creating c_2. As a reaction, the agent asserts the social fact denoting that the items have been shipped. That is, the effect of the behavior would be the satisfaction of commitment c_1. By applying the *Cooperate* pattern, the $customer$ can be provided with those behaviors needed to handle the detachment of c_3 and c_4, as well.

The $merchant$ could also want to withdraw the items from the sale, thereby closing the purchase, for instance if the $customer$ does not react within three days (because it's a limited-time offer). In this case, the commitments will still be in the state Conditional, a fact that we denote through conditional(.). We can model such a behavior by means of the *Withdraw offer* pattern, as follows:

ON threeDaysPassed IF conditional(c_1) $\wedge$ conditional(c_3) $\wedge$ conditional(c_4);
DO cancel(c_1 : C($merchant$, $customer$, c_2, purchaseItemsShipped));
cancel(c_3 : C($merchant$, $customer$, purchaseQuotationRequested, purchaseQuotationProvided));
cancel(c_4 : C($merchant$, $customer$, purchaseItemsPaid, purchaseReceiptDelivered)).

In this case the commitments made by the $merchant$ are no longer needed, and can be canceled.

Let us now take the $customer$'s perspective, which has several possibilities. For instance, being interested in buying a large amount of items, upon creation of c_3, the agent could apply the *Cooperate* pattern (as creditor), and request a quotation.

ON create(c_3) IF interestedInLargeAmount DO assert(purchaseQuotationRequested).

Otherwise, if it is not interested in receiving a quotation (maybe because it's a regular buyer who knows the price in advance), it could simply confirm the purchase, both applying the *Entice* and *Cooperate* patterns w.r.t. the creation of c_2 and subsequent detachment of c_1.

ON create(c_1) IF interestedInItems DO create(c_2).

Also in this case, upon creation of c_2, the $customer$ should be equipped with the behavior(s) needed to handle its detachment.

ON detached(c_2) IF true DO ...$send\ EPO$... assert(purchaseItemsPaid).

The satisfaction of c_2 by $customer$, finally allows the $merchant$ to satisfy its original goal. Conversely, should $customer$ be not interested in the $merchant$'s offer, it could decide to release the latter from its engagement coming from c_1, by applying the *Give up* pattern.

ON create(c_1) $IF\ not$ interestedInItems DO release(c_1).

3.6 The Hiring Process Scenario

We exemplify the proposal in a scenario which is well-known in the field of business processes.

Example 3.2 (Hiring Process [55]). A hirer opens a call for a job position for which many candidates will likely apply over a time period. As long as the position remains open, each candidate is called for an interview, and then evaluated. The evaluation of a single candidate takes time, even weeks. Two processes are involved, namely Hiring Process and Evaluate Candidate. They are synchronized by means of a data store representing the job status in a database. Hiring Process updates the data store when the job is opened, filled or abandoned. Evaluate Candidate queries it immediately upon instantiation. Hiring Process just waits for a message from Evaluate Candidate, indicating that an applicant has been selected for the position, and has accepted the offer. Evaluate Candidate has many instances, each of which carries out the evaluation of a single candidate. Once the job is filled new applicants just receive a position closed message, and any still running instances of evaluate candidate are terminated.

The scenario is an instance of the *one-to-many* pattern of coordination between process instances [24, 55], occurring when the multiplicity of an activity is not in accordance with the multiplicity of another one. In this case, it is necessary to separate them into different interacting processes.

Figure 3.6 shows the solution proposed by Silver, consisting of a *Hiring Process*, each of whose instances tackles a single job opening, an *Evaluate Candidate process*, whose instances tackle each a different candidate, and an *Applicant Process*, whose instances amount to candidates. While the relationship between the instances of *Evaluate Candidate* and *Applicant* is *one-to-one*, the relationship between the instances of the *Hiring Process* and those of the *Evaluate Candidate* process is inherently *one-to-many*. The states of all these processes are to be coordinated and, in particular, the *Hiring Process* must have a way to enable *Evaluate Candidate* when a new job is opened, and to disable its running instances when

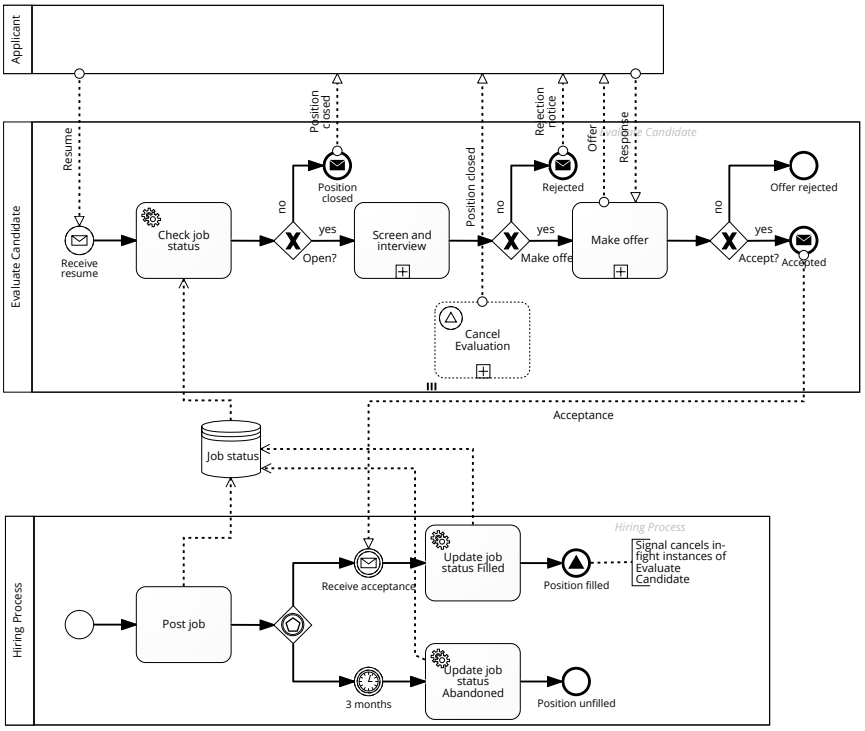

Fig. 3.6: The *Hiring Process* scenario represented in BPMN [55].

the job position has been assigned. To this end, a data storage is introduced, which is external to the processes, but to which all of them have access, supporting synchronization and data consistency. Silver's solution highlights the limitations of BPMN in modeling the coordination, already discussed in Sec. 3.2. In fact, with reference to Fig. 3.6, the BPMN representation only suggests that the *Hiring Process* should update the status of a job opening after receiving the acceptance of an offer by some candidate, and that this will let *Evaluate Candidate* know that it accomplished its task for that opening.

We now explain how to use social commitments to realize the Hiring Process. In Fig. 3.7, hi (hirer), ev_i (evaluator), and i (candidate) represent the agents of this scenario, that observe and use artifacts in a shared environment. For each available position, there will be just one agent playing the hirer role, whereas many evaluators and candidates will be possible. For simplicity, we assume that each candidate i will be evaluated by a specific evaluator ev_i; this does not

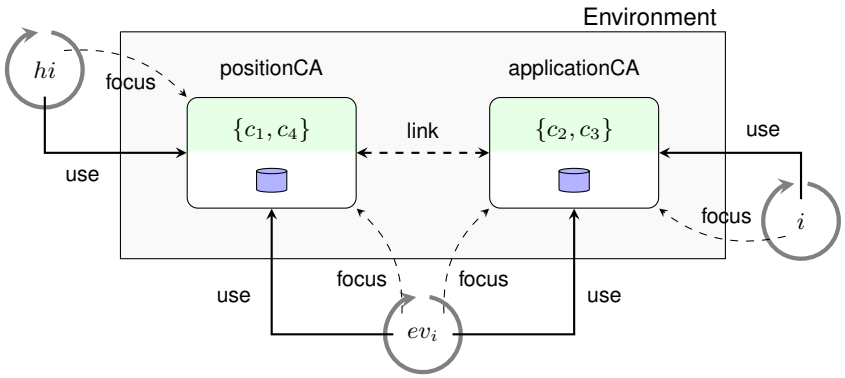

Fig. 3.7: The *Hiring Process* scenario. Commitments $c_1 - c_4$ refer to the set in Fig. 3.8.

exclude, however, that the same agent be an evaluator for different candidates. The workspace includes also two business artifacts extended with social commitments. Each of them stores both the social commitments and the facts that are relevant for the application. The artifact positionCA, in particular, maintains the state of the position, it is instantiated just once, and it is only accessed by the hirer and the evaluators. Instead, the artifact applicationCA is instantiated once for each candidate, each instance keeping track of the status of the application made by that specific candidate.

The Business Artifacts extended with Social Commitments. The normative layer of the two artifacts is expressed by the commitments listed in Fig. 3.8. In particular, c_1 and c_4 will characterize positionCA, while c_2 and c_3 will be maintained inside applicationCA. By operating on the two artifacts agents can make the interaction evolve following the lifecycle of the four commitments. For instance, hi can open the position by creating c_4, thereby enticing an interaction with ev_i. The two artifacts are linked, so the operations performed by the agents make the commitments maintained in both artifacts progress. The meanings of the commitment are as follows.

- *Commitment* c_1. It encodes that ev_i is committed to carry out the evaluation for the application by candidate i according to a predefined procedure. The procedure, outlined in evaluate-candidate$_{ev_i}$, is equivalent to the *Evaluate Candidate* process in Fig. 3.6. The sequence position-filled$_{hi}$ $\cdot$ msg-position-closed$_{ev_i}$, describes that the evaluator informs candidate i that the position is closed as soon as the position is assigned by hi. On the BPMN

c_1 : C$(ev_i, hi,$ post-job$_{hi}$ · apply$_i$, post-job$_{hi}$ · apply$_i$ · evaluate-candidate$_{ev_i}$)

c_2 : C$(ev_i, i,$ post-job$_{hi}$ · apply$_i$, post-job$_{hi}$ · apply$_i$ · inform-outcome$_{ev_i}$)

c_3 : C$(i, ev_i,$ make-offer$_{ev_i}$, make-offer$_{ev_i}$ · (response-yes$_i$ ∨ response-no$_i$))

c_4 : C$(hi, ev_i,$ post-job$_{hi}$ · (accepted$_{ev_i}$ ∨ timeout_3months$_{hi}$), post-job$_{hi}$ · hiring$_{hi}$)

Where:

evaluate-candidate$_{ev_i}$ ≡ position-filled$_{hi}$ · msg-position-closed$_{ev_i}$ ∨
(screen-interview$_{ev_i}$ · (msg-rejection-notice$_{ev_i}$ ∨ make-offer$_{ev_i}$ ·
(response-yes$_i$ · accepted$_{ev_i}$ ∨ response-no$_i$ · offer-rejected$_{ev_i}$)))

inform-outcome$_{ev_i}$ ≡ (msg-position-closed$_{ev_i}$ ∨ msg-rejection-notice$_{ev_i}$ ∨ make-offer$_{ev_i}$)

hiring$_{hi}$ ≡ (accepted$_{ev_i}$ · position-filled$_{hi}$) ∨ (timeout_3months$_{hi}$ · position-abandoned$_{hi}$)

Fig. 3.8: The set of commitments included in the normative layer for the *Hiring Process* scenario.

process, this is equivalent to the *Check Job Status* activity and the consequent message sending in case the position has already been assigned, and it also models the capturing of signal *position filled* sent by the hirer while this evaluator is still processing an application. The rest of the condition, (screen-interview$_{ev_i}$ . . . offer-rejected$_{ev_i}$) encodes all the possible branches in the execution of process *Evaluate Candidate*, including the messages sent to and received from other roles. It is important to note the shape of the antecedent and of the consequent conditions of c_1, among which a temporal relation is captured. Indeed, ev_i's commitment is detached (and hence the agent will have to bring about the evaluation process) only when a job position has been posted, and a candidate has applied for it. In order to model that ev_i is expected to evaluate candidate i only after this has applied for the position, the antecedent condition (i.e., post-job$_{hi}$ · apply$_i$) occurs as a prefix in the consequent condition. A similar pattern is used also in the following commitments.

- *Commitment* c_2. Evaluator ev_i takes also commitment c_2 towards candidate i to take into account the application and to provide i with an answer for the application. Such a commitment has the same antecedent condition of c_1. The answer can either be a message informing that the position has already been closed, a rejection notice, or even an offer for the job. Also in this case, apply$_i$ is used in order to make inform-outcome$_{ev_i}$ follow the satisfaction of the antecedent condition.

- *Commitment* c_3. This commitment is pretty interesting from our point of view. It represents candidate i's promise to answer either "yes" or "no" to an eventual offer made by ev_i. The BPMN in Fig. 3.6 does not specify the internal behavior of the candidate, so an answer for the offer cannot be taken for granted. Indeed, the candidate may never answer; the evaluator would not be able to detect this anomalous situation, and may, thus, await indefinitely. In our opinion, this example highlights the weaknesses of BPMN in modeling the coordination of independent processes. Certainly, one could enrich the evaluator's process so as to wait a predefined time interval for an answer. But this is just a way for handling the exception. With commitments, instead, our major concern is to stimulate the agents to act so as to make the interaction progress. ev_i, for instance, could create c_2 only after the creation of c_3. In this way, the evaluator could legitimately expect an answer, should an offer be made. When candidate i is offered the job, it is stimulated to answer either "yes" or "no" due to the existence of commitment c_3. Any anomalous situations in which the candidate does not answer is clearly detected by the violation of c_3. The evaluator process does not need to capture this eventuality directly.
- *Commitment* c_4. Finally, commitment c_4 represents the engagement of the hirer towards the evaluator, and in particular describes the process carried out by the hirer in Fig. 3.6. The antecedent condition expresses the start event of the process (i.e., post-job$_{ev_i}$) followed by two alternative events that enables the hirer to complete the process. Such a commitment is to be created by the hirer to entice the cooperation of the evaluator as soon as a job is posted. The first event is accepted$_{ev_i}$, meaning that an evaluator has found a suitable candidate. The second event is timeout_3months$_{hi}$, which stands for the complementary event to position-filled$_{ev_i}$. Therefore, after a period of three months, when timeout_3months$_{hi}$ occurs, the position cannot be assigned anymore. The consequent condition of c_4 describes how the hirer completes the process depending on what event has satisfied the antecedent condition. In case of event accepted$_{ev_i}$, hi assign the position and notify this to all the evaluators possibly still running (position-filled$_{ev_i}$). Notably, this allows the evaluators still active to bring about event msg-position-closed$_{ev_i}$ so as to discharge both commitments c_1 and c_2. In case the antecedent has been satisfied by event timeout_3months$_{hi}$, instead, the position status is updated to abandoned.

As an illustration, we now show how it is possible to develop agents amounting to hirer, evaluators, and candidates leveraging the programming patterns presented

in Sec. 3.5. To this end we rely on the 2COMM platform, already introduced in Sec. 3.2, and we discuss a realization of the agents written in JaCaMo[1].

Programming the Agents. JaCaMo extended with 2COMM [10,34] allows implementing agents through the Jason language, as sets of plans expressed as ECA-like rules. In particular, each agent has a *belief base*, a set of ground (first-order) atomic formulas which represent the state of the world according to the agent's vision, and a *plan library*. Moreover, it is possible to specify *achievement* (operator '!') and *test* (operator '?') goals. A Jason plan is specified as $triggering\_event : context \leftarrow body$, where the *triggering_event* denotes the event the plan handles (which can be either the addition or the deletion of some belief or goal), the *context* specifies the circumstances when the plan could is applicable, and the *body* is the course of action that should be taken. With 2COMM, the triggering event can be a state change occurring in some commitment.

We realize the three processes hi, ev_i, and i as Jason agents. Listing 3.1 reports the implementation fo hi. The first plan, at Line 1, means that when hi has the goal of posting a job position (and eventually finding a suitable candidate), it *entices* the cooperation of ev_i by creating commitment c_4. Indeed, condition $\mathsf{accepted}_{ev_i}$, the hirer would like to become true, cannot be met by hi on its own. It needs the cooperation of some evaluator, to take care of the applications. The evaluator will create commitment c_1, and the hirer (being c_1's creditor) will then be in condition to apply the first case of the *cooperate* pattern, by contributing to making the antecedent condition of c_1 become true. This is done by posting a job (Line 3). For detaching the commitment it is necessary that some candidate applies for the position. The last plan, at Line 6, is instead realized by using the second case of the *cooperate* pattern: the hirer, being c_4's debtor, has to properly react as soon as the commitment gets *detached*. So, hi updates the position status depending on the events that have occurred (acceptance or timeout), satisfying the antecedent. It's worth noting that the first and the last plan are strictly linked. The latter allows the agent to satisfy the commitment created in the former, once detached. The absence of the second would make the agent unable to fulfill its engagement, and, thus, liable in case of violation.

The code of the evaluator (see in Listing 3.2) is a bit more complex since it encompasses the whole evaluation process. The first plan, at Line 1, follows both the *cooperate* (as creditor) and *entice* patterns. By reacting to the creation c_4, the agent creates c_1 establishing its availability to perform an evaluation. The second plan, in turn, is triggered when the candidate entices an interaction with the

[1] A running implementation of the *Hiring Process* in JaCaMo, together with the source code of 2COMM and other examples can be found at http://di.unito.it/2comm.

```
+!post-job <- createC4[artifact_id(positionCA)].

+cc(ev_i, hi, post-job · apply, post-job · apply · evaluate-candidate, CONDITIONAL)
  <- postJob[artifact_id(positionCA)].

+cc(hi, ev_i, post-job · (accepted ∨ timeout3months), post-job · hiring, DETACHED)
   : Result
  <- updatePosition(Result)[artifact_id(positionCA)].
```

Listing 3.1: Hirer hi.

```
+cc(hi, ev_i, post-job · (accepted ∨ timeout3months), post-job · hiring, CONDITIONAL)
  <- createC1[artifact_id(positionCA)]

+cc(i, ev_i, make-offer, make-offer · (response-yes ∨ response-yes), CONDITIONAL)
  <- createC2[artifact_id(applicationCA)]

+cc(ev_i, hi, post-job · apply , apply · evaluate-candidate, DETACHED)
  <- screenInterview[artifact_id(applicationCA)];
     // ... Choice; !offerOrReject(Choice).
+!offerOrReject(Choice) : Choice == yes
  <- makeOffer[artifact_id(applicationCA)].
+!offerOrReject(Choice) : Choice == no
  <- rejectionNotice[artifact_id(applicationCA)]; releaseC2.

+responseYes : cc(ev_i, hi, post-job · apply, apply · evaluate-candidate, DETACHED)
  <- offerAccepted[artifact_id(positionCA)]
+responseNo : cc(ev_i, hi, post-job · apply, apply · evaluate-candidate, DETACHED)
  <- offerRejected[artifact_id(positionCA)].

+cc(ev_i, hi, post-job · apply , apply · evaluate-candidate, DETACHED)
   : position-filled | position-abandoned
  <- positionClosed[artifact_id(applicationCA)]; releaseC3.

+position-filled
   : cc(ev_i, i, post-job · apply, post-job · apply · inform-outcome, CONDITIONAL)
  <- cancelC2; releaseC3.
+position-abandoned
   : cc(ev_i, i, post-job · apply, post-job · apply · inform-outcome, CONDITIONAL)
  <- cancelC2; releaseC3.
```

Listing 3.2: Evaluator ev_i.

```
+!find-job <- createC3.

+post-job <- apply[artifact_id(applicationCA)].

+cc(candidate_i, ev_i, make-offer, response-yes ∨ response-no, DETACHED)
  <- // ... Choice; !response(Choice).
+!response(Choice) : Choice == yes
  <- responseYes[artifact_id(applicationCA)].
+!response(Choice) : Choice == no
  <- responseNo[artifact_id(applicationCA)].
```

Listing 3.3: Candidate i.

evaluator by creating c_3. In this case the evaluator reacts creating c_2, so as to encourage the candidate to send an application. The third plan, in which the evaluator reacts to the detachment of commitment c_1 by interviewing the candidate, provides the agent with the ability to handle the evolution of c_1, likely bringing it to satisfaction. To this end, the agent acts upon applicationCA by executing screenInterview. After this operation, the evaluator comes up with a Choice, either accept the candidate or reject it. This choice will, thus, activate a proper behavior (see the plans at Lines 10–13). It is worth noting that, in case of rejection, the agent can also apply the *give up* pattern, by releasing candidate i from its engagement due to c_3. Since an offer won't be made, a response is no longer needed. Plans at Lines 15–18 are, instead, used to react to a candidate's answer, either "yes" or "no", to a possible offer. Accordingly, the evaluator performs an operation on positionCA so as to complete the evaluation process and, then, discharge its commitment c_1. The plan starting at Line 20 captures the situation in which the evaluator is held to inform the candidate as soon as the position gets filled. Interestingly, no specific rule is requested for treating commitment c_2 because whenever such a commitment gets detached, the evaluator satisfies it by satisfying c_1. However, at a normative layer, commitment c_2 is fundamental to detect the misbehavior of the evaluator towards the candidate. Should the position be filled or abandoned with c_2 still conditional, the evaluator can also be equipped with those plans to cancel its commitment no longer needed, following the *withdraw offer* pattern (see Lines 24 and 27).

Finally, Listing 3.3 sketches the pseudocode of a candidate. As soon as the agent has a goal of finding a job, it can create c_3, so as to entice the cooperation of the evaluator. Then, when a job is posted candidate i can decide to send an application (see the plan at Line 3). The last three plans are needed to react to the detachment of commitment c_3 by answering either "yes" or "no" to an offer.

3.7 Conclusions

In this chapter we have addressed one of the challenges in engineering distributed systems: the development of high-level abstractions for modeling interaction. The distribution of goals over a number of interacting and independent execution threads, in fact, demands for special interaction abstractions that allow complex systems to scale up effectively.

In this chapter, we explain the adoption of commitments, which enjoy several interesting properties, for explicitly modeling interaction at a higher level than message passing protocols. Commitments have a normative power, and hence enable agents to create expectations about the behaviors of others. They can be

created and manipulated directly by the agents via a standard set of operations, and this allows agents to reason on how to act upon commitments in order to coordinate with others. Recent proposals, see [56], exploit commitments to enable the design of socio-technical systems, composed of both social (people and organizations) and technical (computers and networks) elements, that satisfy the stakeholders' requirements, and their refinement through design patterns.

We have seen that commitments can be reified in interaction resources, and that they can be manipulated by the agents in autonomy. From a software engineering point of view, there are several advantages. First of all, the modularity is substantially improved since there is a clear separation between the interaction logic, encapsulated within a proper artifact, and the agent logic. This, in turn, promotes software reuse. More importantly, the standard lifecycle of commitments can be the base for developing a number of agent programming patterns. The rationale is that an agent involved in a commitment should possess at least those behaviors for making that commitment evolve toward satisfaction.

Also in the context of industrial applications, multiagent systems proved to be effective in enabling adaptability and flexibility of automated production systems (see, e.g., [57]). Agent platforms are nowadays widely used in industrial domains, ranging from smart environments to logistics. Some of them were developed in industrial settings, as is the case of JADE, which was developed at TIM TILab, the research center of the main Italian telecommunication company. The proposed commitment-based approach, being implemented as a library that can easily be adapted to multiple agent-frameworks, paves the way for a fruitful exploitation of the advantages coming from an explicit representation of social relationships in several application domains.

Although property verification falls outside the scope of this chapter, there are a number of solutions that can be exploited thanks to the use of commitments. Indeed, as a consequence of the induced decoupling, one can verify the correctness of both agents and artifacts in isolation. On the artifact side, it is possible to verify general properties on the commitments that may be possibly created by the agents. For instance, [58] presents the 2CL methodology, an approach for specifying business protocols in a declarative way with social commitments. Being a formal language, 2CL enables several forms of verification. For instance, it is possible to analyze the space of all possible evolutions of a set of commitments in order to determine risky situations. That is, conditions where commitments are likely to be violated. In [42], the authors propose a methodology for checking, at design time, whether a commitment protocol satisfies a predefined set of generic properties such as consistency, effectiveness and robustness. A commitment algebra is presented in [59], where it provides a conceptual basis for reasoning about

protocols in terms of notions like refinement or aggregation, and includes the operators merge and choice and a subsumption relation for protocols.

On the agent side, similar automatic verification techniques can be used to assess the compliance of an agent to a given commitment protocol; see for instance [60]. Moreover, commitments can used as the basis of a dynamic, agent type-checking system which, at execution time, can assess whether an agent is equipped with a set of behaviors that is appropriate for carrying out an interaction [61].

The approach presented in this chapter sets the basis for an important further development: the scaling from commitment to accountability. Accountability has a very broad range of interpretations. Briefly, we can think of accountability as a directed relationship from an account-giver to an account-taker, that results from an agreement between the parties. When an accountability relationships is established, the account-taker can legitimately ask, under some agreed conditions, an account about a process of interest to the account-giver. On the other hand, the account-giver is legitimately required to provide the account to the account-taker. There are several differences between social commitments and accountability. First of all, a social commitment results from the initiative of its debtor and does not require the agreement of its creditor.[2] In addition, social commitments imply *liability*, i.e., when a social commitment is violated the creditor is legitimated to complain against the debtor [63]. However, the creditor has no right to ask for an explanation about the violation. This hampers the transmission of information that the creditor could use to deal with the unexpected situation brought about by the violation. We deem that a computational model of accountability [64–67] is the key to develop feedback/reporting frameworks, similarly to what is often done in human organizations, that can be exploited for handling both anomalous conditions and new opportunities. In [68] an early conceptual view on how accountability can be the means to reach robustness is discussed. Specifically, the idea is that accountability relationships can be used, at design time, to model feedback channels through which information can be conveyed from the agent who produces it, to the agent who can exploit such information in its internal deliberative cycle. This general principle is at the basis of robustness in distributed systems, since it provides the information they need to properly handle a perturbation.

[2]The notion of commitments introduced by [62] does require acceptance from its creditor, but this is not the interpretation we have used in this paper which is aligned with the view presented in [35].

References

[1] M. J. Wooldridge, *Introduction to multiagent systems*. Wiley (2002), ISBN 978-0-471-49691-5.

[2] M. N. Huhns and L. M. Stephens, *Multiagent Systems and Societies of Agents*, chap. 2. MIT Press, ISBN 0262232030, pp. 79–120 (1999), ISBN 0262232030.

[3] R. H. Bordini, L. Braubach, M. Dastani, A. E. Fallah-Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. M. P. O'Hare, A. Pokahr and A. Ricci, A survey of programming languages and platforms for multi-agent systems, *Informatica* **30**, 1, pp. 33–44 (2006).

[4] F. L. Bellifemine, F. Bergenti, G. Caire and A. Poggi, JADE - A Java Agent Development Framework, in *Multi-Agent Programming: Languages, Platforms and Applications*, *Multiagent Systems, Artificial Societies, and Simulated Organizations*, Vol. 15. Springer, pp. 125–147 (2005).

[5] A. Omicini and F. Zambonelli, TuCSoN: a coordination model for mobile information agents, in *Proc. of IIIS'98*. IDI – NTNU, Trondheim (Norway), pp. 177–187 (1998).

[6] F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings and J. Treur, Desire: Modelling Multi-Agent Systems in a Compositional Formal Framework, *Int. J. of Cooperative Information Systems* **06**, 01, pp. 67–94 (1997).

[7] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci and A. Santi, Multi-agent oriented programming with JaCaMo, *Science of Computer Programming* **78**, 6, pp. 747–761 (2013).

[8] I. J. Timm, T. Scholz, O. Herzog, K. Krempels and O. Spaniol, From agents to multiagent systems, in *Multiagent Engineering, Theory and Applications in Enterprises*. Springer, pp. 35–51 (2006).

[9] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Z. Kwiatkowska, J. A. McDermid and R. F. Paige, Large-scale complex IT systems, *Communications of the ACM* **55**, 7, pp. 71–77 (2012), doi:10.1145/2209249.2209268, http://doi.acm.org/10.1145/2209249.2209268.

[10] M. Baldoni, C. Baroglio and F. Capuzzimati, A Commitment-based Infrastructure for Programming Socio-Technical Systems, *ACM Transactions on Internet Technology* **14**, 4, pp. 23:1–23:23 (2014).

[11] M. P. Singh, A social semantics for agent communication languages, in *Issues in Agent Communication*, *Lecture Notes in Computer Science*, Vol. 1916. Springer, pp. 31–45 (2000).

[12] C. Castelfranchi, *Principles of Individual Social Action*. Kluwer, pp. 163–192 (1997).

[13] A. K. Chopra, A. Artikis, J. Bentahar, M. Colombetti, F. Dignum, N. Fornara, A. J. I. Jones, M. P. Singh and P. Yolum, Research directions in agent communication, *ACM Transactions on Intelligent Systems Technology* **4**, 2, pp. 20:1–20:23 (2013), doi:10.1145/2438653.2438655, https://doi.org/10.1145/2438653.2438655.

[14] Y. Demazeau, From interactions to collective behaviour in agent-based systems, in *Proceedings of the 1st. European Conference on Cognitive Science*. Saint-Malo, pp. 117–132 (1995).

[15] D. Weyns, A. Omicini and J. Odell, Environment as a first class abstraction in multiagent systems, *JAAMAS* **14**, 1, pp. 5–30 (2007).

[16] A. Omicini, A. Ricci and M. Viroli, Artifacts in the A&A meta-model for multi-agent systems, *Autonomous Agents and Multi-Agent Systems* **17**, 3, pp. 432–456 (2008).
[17] A. Ricci, M. Piunti and M. Viroli, Environment programming in multi-agent systems: an artifact-based perspective, *Autonomous Agents and Multi-Agent Systems* **23**, 2, pp. 158–192 (2011).
[18] A. Nigam and N. S. Caswell, Business artifacts: An approach to operational specification, *IBM Systems Journal* **42**, 3, pp. 428–445 (2003).
[19] M. Baldoni, C. Baroglio, F. Capuzzimati and R. Micalizio, Commitment-based Agent Interaction in JaCaMo+, *Fundamenta Informaticae* **159**, 1–2, pp. 1–33 (2018).
[20] M. A. Sirbu, Credits and debits on the internet, *IEEE Spectrum* **34**, 2, pp. 23–29 (1997).
[21] Object Management Group (OMG), Business Process Model and Notation (BPMN), Version 2.0, OMG Document Number formal/2011-01-03 (https://www.omg.org/spec/BPMN/2.0/PDF) (2011).
[22] M. P. Singh, Commitments in multiagent systems some controversies, some prospects, in *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi*, chap. 31. College Publications, London, pp. 601–626 (2011).
[23] Object Management Group (OMG), Case Management Model and Notation (CMMN), Version 1.1, OMG Document Number formal/2016-12-01 (https://www.omg.org/spec/CMMN/1.1/PDF) (2016).
[24] M. Dumas, On the convergence of data and process engineering, in *Proc. of Advances in Databases and Information Systems*, *LNCS*, Vol. 6909. Springer, pp. 19–26 (2011).
[25] M. Baldoni, C. Baroglio, F. Capuzzimati and R. Micalizio, Objective Coordination with Business Artifacts and Social Engagements, in *Business Process Management Workshops*, *Lecture Notes in Business Information Processing*, Vol. 308. Springer, pp. 71–88 (2018).
[26] K. Bhattacharya, R. Hull and J. Su, *A data-centric design methodology for business processes*, Handbook of Research on Business Process Modeling. IGI Publishing, pp. 503–531 (2009).
[27] A. Omicini and S. Ossowski, Objective versus subjective coordination in the engineering of agent systems, in *AgentLink*, *LNCS*, Vol. 2586. Springer, pp. 179–202 (2003).
[28] F. Zambonelli, N. R. Jennings and M. Wooldridge, Developing multiagent systems: The Gaia methodology, *ACM Trans. Softw. Eng. Methodol.* **12**, 3, pp. 317–370 (2003).
[29] M. Wooldridge, N. R. Jennings and D. Kinny, The GAIA methodology for agent-oriented analysis and design, *Autonomous Agents and multi-agent systems* **3**, 3, pp. 285–312 (2000).
[30] M. de Brito, J. F. Hübner and O. Boissier, A conceptual model for situated artificial institutions, in *Computational Logic in Multi-Agent Systems*. Springer, pp. 35–51 (2014).
[31] R. H. Bordini, J. F. Hübner and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons (2007), ISBN 0470029005.
[32] A. Ricci, M. Piunti, M. Viroli and A. Omicini, *Environment Programming in CArtAgO*. Springer US, Boston, MA, ISBN 978-0-387-89299-3, pp. 259–288 (2009), ISBN 978-0-387-89299-3.

[33] J. F. Hubner, J. S. Sichman and O. Boissier, Developing organised multiagent systems using the MOISE+ model: Programming issues at the system and agent levels, *Int. J. Agent-Oriented Softw. Eng.* **1**, 3/4, pp. 370–395 (2007).

[34] M. Baldoni, C. Baroglio, R. Micalizio and S. Tedeschi, Programming agents by their social relationships: A commitment-based approach, *Algorithms* **12**, 4, p. 76 (2019).

[35] M. P. Singh, An ontology for commitments in multiagent systems, *Artif. Intell. Law* **7**, 1, pp. 97–113 (1999).

[36] M. Dastani, D. Grossi, J. C. Meyer and N. A. M. Tinnemeier, Normative multi-agent programs and their logics, in *Normative Multi-Agent Systems, 15.03.–20.03.2009*, *Dagstuhl Seminar Proceedings*, Vol. 09121. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2009).

[37] M. P. Singh, Distributed Enactment of Multiagent Workflows: Temporal Logic for Web Service Composition, in *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14–18, 2003, Melbourne, Victoria, Australia, Proceedings*. ACM, pp. 907–914 (2003).

[38] E. Marengo, M. Baldoni, C. Baroglio, A. Chopra, V. Patti and M. Singh, Commitments with regulations: reasoning about safety and control in REGULA, in *Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, Vol. 2, pp. 467–474 (2011).

[39] P. R. Telang and M. P. Singh, Specifying and Verifying Cross-Organizational Business Models: An Agent-Oriented Approach, *IEEE Trans. Services Computing* **5**, 3, pp. 305–318 (2012).

[40] P. R. Telang, M. P. Singh and N. Yorke-Smith, Relating Goal and Commitment Semantics, in *Post-proc. of ProMAS*, *LNCS*, Vol. 7217. Springer (2011).

[41] S. Rafaeli, *Sage Annual Review of Communication Research: Advancing Communication Science: Merging Mass and Interpersonal Processes*, chap. (Chapter 4) Interactivity: From new media to communication. Sage (1988).

[42] P. Yolum and M. P. Singh, Commitment Machines, in *Intelligent Agents VIII, 8th Int. WS, ATAL 2001*, *LNCS*, Vol. 2333. Springer, pp. 235–247 (2002).

[43] N. Fornara and M. Colombetti, Defining Interaction Protocols using a Commitment-based Agent Communication Language, in *Proc. of the Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2003)*. ACM, pp. 520–527 (2003).

[44] A. K. Chopra, *Commitment Alignment: Semantics, Patterns, and Decision Procedures for Distributed Computing*, Ph.D. thesis, North Carolina State University, Raleigh, NC (2009).

[45] P. Torroni, F. Chesani, P. Mello and M. Montali, Social Commitments in Time: Satisfied or Compensated, in *Declarative Agent Languages and Technologies VII, 7th International Workshop (DALT 2009)*, Vol. 5948, pp. 228–243 (2010).

[46] M. El-Menshawy, J. Bentahar and R. Dssouli, Verifiable semantic model for agent interactions using social commitments, in *Languages, Methodologies, and Development Tools for Multi-Agent Systems, Second International Workshop, LADS 2009, Torino, Italy, September 7–9, 2009, Revised Selected Papers*, no. 6039 in Lecture Notes in Computer Science. Springer, pp. 128–152 (2009).

[47] M. Baldoni, C. Baroglio, E. Marengo and V. Patti, Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach, *ACM Trans. on*

Intelligent Sys. and Tech., Special Issue on Agent Communication **4**, 2, pp. 22:1–22:25 (2013).

[48] M. Philippsen, A survey of concurrent object-oriented languages, *Concurrency - Practice and Experience* **12**, 10, pp. 917–980 (2000).

[49] M. Schumacher, *Objective Coordination in Multi-agent System Engineering: Design and Implementation*. Springer-Verlag, Berlin, Heidelberg (2001), ISBN 3-540-41982-9.

[50] M. R. Zatelli, A. Ricci and J. F. Hübner, Integrating interaction with agents, environment, and organisation in JaCaMo, *IJAOSE* **5**, 2/3, pp. 266–302 (2016).

[51] O. Boissier, R. H. Bordini, J. F. Hübner and A. Ricci, Dimensions in programming multi-agent systems, *The Knowledge Engineering Review* **34**, p. e2 (2019).

[52] P. Yolum and M. P. Singh, Flexible protocol specification and execution: applying event calculus planning using commitments, in *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15–19, 2002, Bologna, Italy, Proceedings*. ACM, pp. 527–534 (2002).

[53] M. Winikoff, W. Liu and J. Harland, Enhancing commitment machines, in *Declarative Agent Languages and Technologies II, Second International Workshop, DALT 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers*, no. 3476 in Lecture Notes in Computer Science. Springer, pp. 198–220 (2004).

[54] P. Telang, M. Singh and N. Yorke-Smith, A coupled operational semantics for goals and commitments, *Journal of Artificial Intelligence Research* **65**, pp. 31–85 (2019).

[55] B. Silver, *BPMN Method and Style, with BPMN Implementer's Guide*, 2nd edn. Cody-Cassidy Press, Aptos, CA, USA (2012).

[56] Ö. Kafali, N. Ajmeri and M. P. Singh, DESEN: specification of sociotechnical systems via patterns of regulation and control, *ACM Trans. Softw. Eng. Methodol.* **29**, 1, pp. 7:1–7:50 (2020).

[57] VDI, VDI/VDE 2653 Blatt 1 - Multi-agent systems in industrial automation - Fundamentals, Tech. Rep., VDI - The Association of German Engineers - Department of Industrial Information Technology (2018).

[58] M. Baldoni, C. Baroglio, E. Marengo, V. Patti and F. Capuzzimati, Engineering commitment-based business protocols with the 2CL methodology, *Autonomous Agents and Multi-Agent Systems* **28**, 4, pp. 519–557 (2014).

[59] A. U. Mallya and M. P. Singh, An algebra for commitment protocols, *Auton. Agents Multi Agent Syst.* **14**, 2, pp. 143–163 (2007).

[60] D. D'Aprile, L. Giordano, A. Martelli, G. L. Pozzato, D. Rognone and D. Theseider Duprè, *Information Systems: Crossroads for organization, management, accounting and engineering*, chap. Business process compliance verification: An annotation based approach with commitments, pp. 563–570 (2012).

[61] M. Baldoni, C. Baroglio, F. Capuzzimati and R. Micalizio, Type Checking for Protocol Role Enactments via Commitments, *Journal of Autonomous Agents and Multi-Agent Systems* **32**, 3, pp. 349–386 (2018).

[62] C. Castelfranchi, Commitments: From Individual Intentions to Groups and Organizations, in *Proceedings of the First International Conference on Multiagent Systems (ICMAS)*. The MIT Press, San Francisco, California, USA, pp. 41–48 (1995).

[63] A. K. Chopra and M. P. Singh, From social machines to social protocols: Software engineering foundations for sociotechnical systems, in *Proc. of the 25th Int. Conf. on WWW* (2016).

[64] M. Baldoni, C. Baroglio, K. M. May, R. Micalizio and S. Tedeschi, Computational Accountability, in *Deep Understanding and Reasoning: A challenge for Next-generation Intelligent Agents, URANIA 2016*, Vol. 1802. CEUR, Workshop Proceedings, pp. 56–62 (2016).

[65] M. Baldoni, C. Baroglio, O. Boissier, K. M. May, R. Micalizio and S. Tedeschi, Accountability and Responsibility in Agents Organizations, in *PRIMA 2018: Principles and Practice of Multi-Agent Systems*, no. 11224 in Lecture Notes in Computer Science. Springer, pp. 261–278 (2018).

[66] M. Baldoni, C. Baroglio, K. M. May, R. Micalizio and S. Tedeschi, MOCA: An ORM MOdel for Computational Accountability, *Journal of Intelligenza Artificiale* **13**, 1, pp. 5–20 (2019a).

[67] M. Baldoni, C. Baroglio, O. Boissier, R. Micalizio and S. Tedeschi, Engineering Business Processes through Accountability and Agents, in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13–17, 2019*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1796–1798 (2019b).

[68] M. Baldoni, C. Baroglio and R. Micalizio, Fragility and Robustness in Multiagent Systems, in *To appear in the 8th International Workshop on Engineering Multi-Agent Systems (EMAS 2020) @ AAMAS 2020, 8–9 May 2020, Auckland, New Zealand* (2020).