Commitment-based Agent Interaction in JaCaMo+

(Article begins on next page)

27 April 2024

Commitment-based Agent Interaction in JaCaMo+

Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, Roberto Micalizio

Università degli Studi di Torino, Dipartimento di Informatica

c.so Svizzera 185, I-10149 Torino (Italy)

firstname.lastname@unito.it

Abstract. We present the JaCaMo+ framework for programming multiagent systems (MAS), where agents interact thanks to commitment-based interaction protocols. Commitment protocols are realized as artifacts that maintain a social state and notify to the participating agents those events that are relevant to the interaction. We discuss the advantages, like increased modularity and flexibility, that are brought by commitment-ruled interactions with respect to other proposals. We trace back such advantages to the possibility of relying on a standardized commitment lifecycle. We explain how to use the framework to program interacting agents by using the Netbill protocol as running example, and the Gold Miners scenario as a more complex programming example.

Keywords: Interaction-Oriented Programming, Commitment-based protocols, JaCaMo

1. Introduction

Many researchers claim that an effective way to approach the design and development of a MAS consists in conceiving it as a structure composed of four main entities: *Agents*, *Environment*, *Interactions*, and *Organization* (AEIO) [70, 28, 38]. Such a separation of concerns enjoys many advantages from a software engineering point of view, since it enables a modular development of code that eases code reuse and maintainability. For what concerns interaction, since the late '80s, studies on distributed artificial intelligence, studies on formal theories of collective activity, team, or group work, and studies on cooperation implicitly identified in *commitment* the glue of group activity: commitments link the actions of the group members and the group members with each other [17, 55, 46]. In particular, *social commitments* [56] are a kind of social relationship with a normative value, that makes it possible for the agents to have

Address for correspondence: Matteo Baldoni, Università degli Studi di Torino, Dipartimento di Informatica, c.so Svizzera 185, I-10149 Torino (Italy)

expectations on one another and coordinate their activities. A social commitment models the directed relation between two agents, a *debtor* and a *creditor*; it is created by the debtor, it is explicitly manipulated by the agents, and progresses according to a well-defined lifecycle along with the agents' activities [61]. Commitment-based interaction protocols [67, 68, 69] have a declarative nature which allows to capture the contractual relationships among the concerned partners in a natural way, avoiding to encode some strict order in which messages should be exchanged — although orders can be expressed if necessary [42, 9]. This kind of protocols allows flexibility in agents' enactments; this, in turn, enables agents to profit of unforeseen opportunities and to better face unpredictable situations. These characteristics make commitment protocols particularly fit to cross-organizational application domains, where only a minimal set of constraints can often be specified to make parties interact. In such contexts, in fact, there is typically no authority that can prescribe specific ways for executing activities [29, 9].

Currently, there are many frameworks that support designers and programmers in realizing one of the AEIO components (e.g., [11, 14, 49, 15, 62, 12]). JaCaMo [12] is, to the best of our knowledge, the most complete among the well-established programming frameworks, providing a thorough integration of agents, environments, and organizations into a single platform. JaCaMo, however, lacks integration of interaction as a first-class component, being current solutions mostly ad hoc. We discuss them, and in particular [70], in the next section. This paper fills the gap by presenting the JaCaMo+ platform. JaCaMo+ agents engage commitment-based interactions which are reified as JaCaMo+ artifacts. Such artifacts represent the interaction social state and provide the roles JaCaMo+ agents enact. They can also be used to implement monitoring functionalities, for verifying that the on-going interactions respect the commitments and for detecting violations and violators.

Contributions.   The main contributions of this work are: *(i)* the proposal and motivation of a commitment-based interaction component; *(ii)* the conceptual model of the JaCaMo+ framework, as a realization of the MERCURIO proposal [10, 2]; *(iii)* the description and explanation of the JaCaMo+ implementation and how is it used for realizing multiagent systems; *(iv)* a description of the Gold Miners as realized in JaCaMo+ by exploiting commitment-based interaction components.

Organization.   The paper is organized as follows. Section 2 introduces commitment-based interaction components, providing motivations to their development and the conceptual model of our proposal. A commitment-based representation of the Netbill interaction protocol is introduced and used as running example. Section 3 explains JaCaMo+ and its main characteristics. Section 4 describes, as a more consistent application, a JaCaMo+ realization of Gold Miners, used as programming challenge at CLIMA VII. Section 5 compares interaction as realized in JaCaMo+ with other solutions and discusses the advantages of the proposal. Conclusions end the paper.

2.   A Commitment-based Interaction Component

JaCaMo [12] is a platform integrating Jason as an agent programming language, CArtAgO as a realization of the A&A meta-model [65], and $Moise^+$ as a support to the realization of organizations [41]. In this section we motivate the choice of relying on JaCaMo, and we describe how JaCaMo+ is obtained by integrating a first-class component for commitment-based interaction protocols inside JaCaMo. The JaCaMo platform stems from other proposals, like [26, 64], as it allows realizing multiagent systems that

not only involve many autonomous entities, but where such entities interact in complex ways by way of social structures and norms that regulate the overall social behavior, and where a shared environment is an important coordination means for the agents. Indeed, JaCaMo is currently the only platform that integrates three fundamental aspects of multiagent system programming, namely agent programming, environment programming, and organization programming, thus realizing the AIEO model [28] almost completely. Synergies between the three dimensions bring about many benefits among which:

- the repertoire of the agents actions is dynamic because it depends on the available artifacts;

- actions gain a process-based semantics, which makes it possible to define long-term actions as well as coordinating actions;

- an explicit and well-defined notion of success/failure for actions is provided;

- the interaction between agents and organizations is uniformly obtained using the same mechanisms that enable agent-artifact interaction;

- organizations can be reshaped dynamically by acting on artifacts;

- the delegation of organizational goals to agents is facilitated.

Works like [50, 37, 43], that apply the platform in various contexts, and works like [54, 70] that extend its functionalities, prove the interest it raises and its success.

2.1.   Interaction and Commitment-based Protocols

Leaving aside the recent proposal in [70], that we discuss below, JaCaMo still lacks a synergistic integration of the fourth main dimension envisaged in [28], i.e., interaction. Traditionally, in JaCaMo interaction is realized either by way of ad hoc direct communication between agents, based on Jason speech acts [14, Chapter 6], or by relying on communication and coordination artifacts [53, 12]. The advantages of the integration of interaction as a first-class component of JaCaMo would be many [70]. For instance, it would allow decoupling the interaction code from the agent code, and this would in turn increase the maintainability of the software with respect to solutions where the interaction code is distributed and immersed in the agent programs [31]. It would facilitate the re-use of the interaction code, the design and composition of interaction protocols, and their validation [44, 60]. It would foster the openness of multiagent systems because it would facilitate agents in joining/leaving systems of interacting agents at run-time [35]. The proposal in [70] is the first that is aimed at integrating interaction in JaCaMo as a first-class component of the system. The described *interaction component* 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). The choice of relying on automata is well-supported in the literature (see, e.g., [47, 16, 32]) but, as [69, 66] point out, such protocol specifications show a rigidity that prevents agents

from taking advantage from opportunities and from handling exceptions in dynamic and uncertain multiagent environments, as JaCaMo MASs could likely be. Agents are, in fact, confined to the execution sequences provided by the automaton. In contrast to this approach, since the seminal papers by Yolum and Singh [67, 68, 69], commitment-based protocols have been raising a lot of attention, see for instance [35, 20, 63, 33, 9]. Protocol actions affect the state of the system, which consists both of the state of the world and also of the *commitments* that agents have made to each other. Commitments motivate agents to perform their next actions. This happens because agents want to comply with the protocol and provide what promised to the other parties. Another key feature of commitment protocols is their *declarative nature*, which allows to naturally capture the contractual relationships among the partners rather than strictly encoding the order in which messages should be exchanged. Whatever action an agent decides to perform is fine if they accomplish their commitments, satisfying the expectations they have on one another. The proposal in [70], though well-integrated with the agent, environment, and organization dimensions of JaCaMo, does not show this same flexibility. Agents are solicited to act by obligations created *externally* to the agents, in a way that resembles the *call-back* mechanism. Agents can comply their obligations along with the execution of other activities of their own, as well as they may decide to not satisfy some obligation that was put on them. However, they would not be free to execute the protocol steps in a different order, as it may, for instance, be necessary in order to adapt to exceptional conditions in the environment or to involve new agents who just entered the system. Another issue is that the *rationale* by which obligations are created is not available to the agents in a form that can be reasoned about: the social meaning of the protocol steps and of the obligations is only implicitly encoded inside the protocol. Also this aspect has an impact on flexibility because it reduces the capacity of the agents to deliberate about their own behavior. In fact, works such as [24, 27] show the importance, for the agents to have full control of their conduct, to enable reasoning about the social consequences of their actions by exploiting constitutive norms that link the agents' actions to their respective social meanings. The last aspect to consider is that interaction is *not bound* to be a procedure inside some organization. In some cases interaction is *among agents* and each agent decides what is best for itself; in other cases guidelines amount to declarative, underspecified constraints that leave agents the freedom to take strategic decisions about their behavior.

We realize interaction based on *social commitments*, intended as first-class objects that can be used for agent programming. A social commitment [56, 59] $C(x, y, s, u)$ 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 [26] for social relationships among autonomous parties. Unlike obligations, commitments are manipulated by agents through the standard operations *create*, *cancel*, *release*, *detach*, *satisfy*, *discharge*, *expire*, *violate*, *assign*, *delegate*. Part of these are implicit and occur simultaneously with the events by which the antecedent becomes true (detach), the antecedent becomes false (expire), the consequent becomes true (satisfy), or the consequent becomes false (violate). The other operations are explicitly executed either by the creditor (release, by which a commitment is removed, and assign, by which the creditor is changed) or by the debtor (create, by which the commitment is created, cancel, by which it is removed, and delegate, by which the debtor is changed). Commitment evolution follows the lifecycle formalized in [61], which is reported in Figure 1. A commitment is *Violated* either when its antecedent is true but its consequent will forever be false, or when it is canceled when Detached. It is *Satisfied*, when the engagement is accomplished. It is *Expired*, when it is no longer in effect and therefore the debtor would
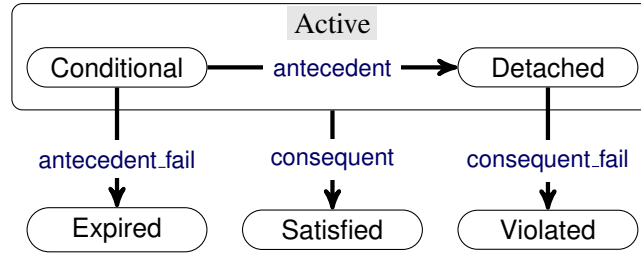
Figure 1.   Commitment life cycle[61].

not fail to comply even if does not accomplish the consequent. A commitment should be *Active* when it is initially created. Active has two substates: *Conditional* as long as the antecedent does not occur, and *Detached* when the antecedent has occurred. 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. Agents, however, are norm-autonomous [22]: indeed, not only they decide whether satisfying the obligation entailed by the commitment, once detached, they also become debtors by their own decision. In other words, a commitment is taken by a debtor towards a creditor on its own initiative. An agent creates commitments towards other agents while it is trying to achieve its goals (or precisely to the aim of achieving its goals) [61]. 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 [61]. 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* [51]. 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" [59, Sect. 4.4]. On this foundation, 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.

   Commitment-based interaction protocols [57, 68, 67] are interaction patterns given in terms of commitments, involving a set of predefined *roles*. They assume that a (notional) *social state* is available and inspectable by all the involved agents. The social state traces which commitments currently exist between any two agents, and the states of these commitments according to the lifecycle, together with other literals that are relevant to their interaction [66]. Agents modify the social state by executing protocol actions that are defined in terms of updates to the social state (e.g. add a new commitment, release another agent from some commitment, satisfy a commitment). A commitment protocol is a *set of actions*, involving the foreseen roles, and whose semantics is agreed upon by all of the participants [68, 67, 20].

Let $\mathcal{B}$ be a nonempty set of events. Let $\mathcal{E}$ be the set of event temporal expressions in precedence logic [58], generated from $\mathcal{B}$. Let ρ be a set of role names. Let $\mathcal{C}$ be the set of possible commitments $C(x, y, q, p)$, where $q, p \in \mathcal{E}$ and where $x, y \in \rho$. Let $\mathcal{Q}$ be the set of possible operations on commitments.

Definition 2.1. A commitment protocol is a tuple $\langle A, \rho, pow, I \rangle$, where A is the set of protocol actions, ρ is a set of role names, $pow : \rho \to 2^A$ is a mapping that associates each role with the set of actions empowered to that role, and I is a set of commitments that hold at the beginning of the interaction. Each action $a \in A$ is a triple $\langle n, E, C \rangle$ where n is the action name, $E = \{e \mid e \in \mathcal{Q}\}$ is the social meaning of the action, and C is a condition to be verified on the social state, specifying the context in which the action yields the social meaning. For each role name $x \in \rho$, $pow(x)$ denotes the subset of protocol actions in A that any player of the role with name x can perform in P. So, $pow(x)$ is the set of powers that are endowed to any agent enacting x.

So, protocols define name spaces that are assumed to be known and shared by the interacting agents, in particular, they define role names and action names. Contexts can be used to sequentialize actions, when needed.

Given a finite set of agents $\mathcal{A}$ and a commitment protocol P, an *enactment* of the protocol is given by $\langle P, \mathcal{A}, play \rangle$ where the function $play : P.\rho \to 2^{\mathcal{A}}$ associates to each role of the commitment protocol the set of agents playing that role. Such agents will be empowered with the actions associated to the role. An *interaction artifact* amounts to a *commitment protocol enactment* together with the *social state* of the interaction.

Example 2.2. (Netbill Protocol)
The Netbill protocol allows a customer to buy a product from a merchant. We rely on the description in [69, 66]. The protocol involves two roles, *customer* and *merchant*. When an agent, playing *customer*, requests a quote from an agent playing *merchant*, the merchant sends the quote. If the customer accepts the quote, the merchant sends the goods, then waits for the payment in the form of an electronic payment order. It is assumed that the goods cannot be used until the merchant has sent the decryption key. Once a customer has sent payment, the merchant will send the decryption key along with a receipt. Table 1 shows the Netbill protocol as a commitment-based protocol. For each of the two roles, the table reports the actions it can perform. Each action affects the social state in the way reported next to the action. For instance, when the merchant executes *sendQuote*, the social state is modified by the creation of two commitments, namely *C(merchant, customer, acceptedQuotation(Item, Price), goods)* and *C(merchant, customer, paid, receipt)*. The fact *quotation(Item, Price)* is also recorded in the social state. On the other hand, sending a receipt makes sense only in a context in which payment already occurred. It is worth noting that the social meaning of an action may comprise expressions like *quotation(Item, Price)* or *acceptedQuotation(Item, Price)*. Here, *Item* and *Price* are not variables, i.e. they are not asserted as variables in the social state depending on some state of condition, they are *formal parameters*, and will be substituted by actual parameters (ground terms) when the action is executed. Therefore, *Item* and *Price* will be bound at runtime to particular values, for example *"Good-1"* and *1000*. In other words, the agent will execute the action *sendQuote("Good-1", 1000)*. The scope of formal parameters is always and only the definition of the social effect of the action. At runtime, social effects will always have their parameters replaced by actual values.

Action	Role	Meaning	Context
sendRequest(Item)	*customer*	"add requestedQuote(Item)"	true
sendQuote(Item, Price)	*merchant*	create(*C(merchant, customer, acceptedQuotation(Item, Price), goods))*, create(*C(merchant, customer, paid, receipt))*, "add quotation(Item, Price)"	true
sendAccept(Item, Price)	*customer*	create(*C(customer, merchant, goods, paid)*, "add acceptedQuotation(Item, Price)"	quotation(Item, Price)
sendReject(Item, Price)	*customer*	release(*C(merchant, customer, acceptedQuotation(Item, Price), goods))*, release(*C(merchant, customer, paid, receipt))*, "add rejectedQuotation(Item, Price)"	quotation(Item, Price)
sendGoods	*merchant*	create(*C(merchant, customer, paid, receipt))*, "add goods"	true
sendEPO	*customer*	"add paid"	true
sendReceipt	*merchant*	"add receipt"	paid

Table 1. The Netbill protocol: actions, roles that can execute them, the social meaning of actions, and the contexts in which action execution yields its social meaning.

2.2. Conceptual Model

Figure 2 shows how in our proposal the interaction dimension relates to the other three dimensions of JaCaMo. The agent, the organization, and the environment dimensions are reported as they are presented in [12]. The main element is the *Interaction Artifact*. It extends the environmental notion of *Artifact*, thus an agent can create, dispose and manipulate an interaction artifact in the same way it can do with any artifact. An Interaction Artifact results as the composition of a *Commitment Protocol* and a *Social State*. Agents can use the *Actions* provided by a Commitment Protocol to interact. Data exchanged along the interaction, and that are relevant to the interaction itself, are stored as observable properties, thus realizing a kind of mediated communication [10].

The protocol actions are partitioned based on *Roles*. A Role represents the interface between a Commitment Protocol and an agent who decides to participate in an enactment of such a protocol. In other words, an agent can join (or create) a Commitment Protocol enactment only by playing a *Role* that the protocol defines. A Commitment Protocol is composed of, at least, one role; depending on the role cardinality, multiple agents may play a same role. An action can only be executed by agents
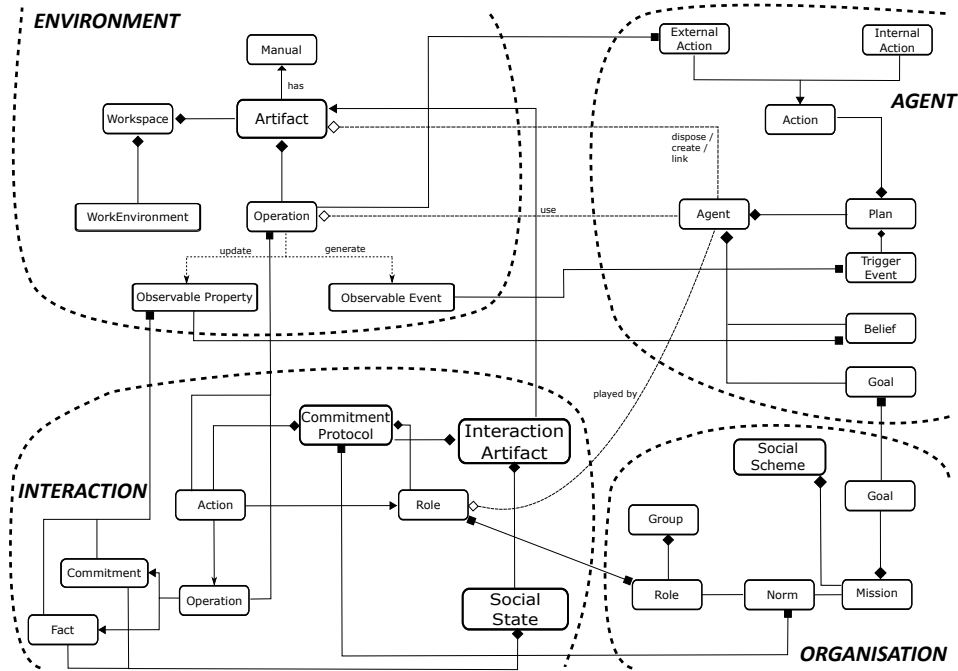
Figure 2.   An overview of the integration among interaction artifacts and JaCaMo components[12].

which are playing the role associated with that action. The execution of an action modifies the state of the interaction. In particular, it may create new commitments or make existing commitments evolve according to the commitment lifecycle. The concept of Action is mapped on the *Operation* element of the environment dimension. The execution of a protocol action is analogous to the execution of an Operation. The difference stands in the particular type of artifact, that in our case will be an Interaction Artifact, whose operations implement the actions, that are provided by a Commitment Protocol. An Action is defined in terms of the consequences its execution causes, i.e. one or more basic operations on Commitments and/or Facts, like the assertion of a fact, the detachment of a commitment, the satisfaction of a commitment. Both Commitments and Facts can be seen as *observable properties* in the environment dimension. Thus, in general, the execution of a commitment protocol action will cause either the update of some observable properties, or the generation of some *observable events* (for instance, a commitment becomes Satisfied, or Violated), or the assertion of a fact. Events can trigger Agent's plans.

The set of current commitments and facts, belonging to an ongoing interaction, is stored in and maintained by the *Social State*. This is not directly accessible to the agents, who will use the Interaction Artifact only by means of the provided role Actions. Notice that, the JaCaMo integration of Agents and Artifacts supports the direct mapping of observable properties into the agents' belief bases. This means that an agent, participating into a commitment-based interaction, enjoys the extension of its belief base with facts and commitments provided by the Interaction Artifact. Modifications to the Social State are notified as Observable Events.

The organisational dimension of JaCaMo prescribes global *goals* that agents, participating into the

organisation, must achieve by enacting organisational *roles*. Their tasks, i.e. organisational goals, are structured in *missions*, and become private goals in the agent dimension. Each mission is composed of, at least, one organizational goal. *Norms* define obligations and permissions for roles, i.e. an agent playing a role is obliged or permitted to commit to a mission [12]. The interaction dimension has an intrinsic organisational nature: similarly to norms, commitments express a kind of normative regulation, while commitment protocols are, actually, sets of constitutive norms by which protocol actions are given their social meaning. This proximity is depicted in Figure 2 by connecting the two concepts with a bidirectional conceptual link. Likewise, interaction roles and organisational roles share the same nature: they represent normative outfits, adopted by the agents which respectively act in the context of an Interaction Protocol or of an Organisation. In JaCaMo, however, it is not possible to express direct agent-to-agent normative relationships: norms are always related to missions, and, therefore, they are not under the control of the obliged agent. The interaction dimension of JaCaMo+ provides suitable means for adopting commitments as the abstraction of such relationships. Commitment protocols are directly used by agents to coordinate their activities, for achieving their (private or organisational) goals when they are unable to fulfill them autonomously. In other words, the organisational dimension concerns global goals assigned to agents, while the interaction dimension provides normative interaction patterns that agents can leverage to achieve goals.

3. JaCaMo+

This section explains how the interaction component is provided by JaCaMo+, how it works, and how it is integrated with Jason agents. Briefly, interaction artifacts reify the execution of commitment-based protocols, they include the social state of the interaction, and they enable Jason agents to be notified about the relevant events that occur in the social state. Since an artifact is a programmable, active entity, it can act as a monitor of the interaction. The artifact can therefore detect violations that it can ascribe to the violator without any need of agent introspection.

3.1. Interaction Component

A JaCaMo+ interaction artifact encodes a commitment protocol, that is structured into a set of *roles*, agents can enact. By enacting a role, an agent gains the rights to perform *social actions*, whose execution has public social consequences, expressed in terms of commitments. If an agent tries to execute an action which is not associated with the role it is enacting, the artifact raises an exception that is notified to the violator. On the other hand, when an agent performs a protocol action that pertains to its role, the social state is updated accordingly, for example, by adding new commitments, or by modifying the state of existing commitments. Interaction artifacts are implemented by the class `ProtocolArtifact`, which extends CArtAgO artifacts. Such a class provides basic manipulation methods for managing the social state: commitment creation and update, creation of facts, notification of events to agents that focused on the artifact. An interaction artifact provides the roles of the protocol and the actions that are associated to each role. As an example, the following code defines the roles of the Netbill protocol (Example 2.2).

```
1 public class NetbillProtocol extends ProtocolArtifact {
2    public static String ARTIFACT_TYPE = "Netbill";
3    public static String MERCHANT_ROLE = "merchant";
4    public static String CUSTOMER_ROLE = "customer";
5    static {
```

```
6        addEnabledRole(MERCHANT_ROLE, Merchant.class);
7        addEnabledRole(CUSTOMER_ROLE, Customer.class);
8      }
9      ...
10  }
```

A protocol role is implemented by extending the class `PARole`, provided by the class `ProtocolArtifact`. `PARole` can also be directly enacted by agents willing to inspect the social state of the interaction, i.e. to retrieve facts and commitments currently holding in the social state. It is implemented as an *inner* class of `ProtocolArtifact`. Protocol designers should define roles as subclasses of `PARole` and as inner classes of the protocol class:

```
1  public class NetbillProtocol extends ProtocolArtifact {
2    ...
3    public class Merchant extends PARole {
4      public Merchant(String playerName, IPlayer player) {
5        super(MERCHANT_ROLE, player);
6      }
7      ... actions ...
8    }
9    public class Customer extends PARole {
10     public Customer(String playerName, IPlayer player) {
11       super(CUSTOMER_ROLE, player);
12     }
13     ... actions ...
14   }
15 }
```

In CArtAgO, the Java annotation[1] *@OPERATION* marks a public operation that agents can invoke on the artifact. In JaCaMo+, a method tagged with *@OPERATION* corresponds to a protocol action. We introduced the annotation *@ROLE* to specify, when necessary, which roles are enabled to use that particular action. For instance, the next listing reports the implementation of the action *sendQuote*, whose effects on the social state are defined by means of primitives for commitment manipulation and social fact addition. After execution, the updates to the social state will be automatically notified to the focusing agents.

```
1  @OPERATION
2  @ROLE(roleName=MERCHANT_ROLE)
3  public void sendQuote(String item, String price) {
4    RoleId merchant = getRoleIdByPlayerName(getOpUserName());
5    RoleId customer = getRoleIdByGenericRoleName(CUSTOMER_ROLE).get(0);
6    try {
7      createCommitment(new Commitment(merchant, customer,
8          new Fact("acceptedQuotation", item, price), "goods"));
9      createCommitment(new Commitment(merchant, customer, "paid", "receipt"));
10     assertFact(new Fact("sendQuote", item, price, merchant.toString()));
11     assertFact(new Fact("quotation", item, price, merchant.toString()));
12   } catch (MissingOperandException e) { ... handle exception ... }
13 }
```

Listing 1.   Excerpt of the sendQuote protocol action.

Indeed, the interaction artifact maintains an *explicit representation of the social state*. By *focusing* on an artifact, an agent registers to be notified of events that are generated inside the artifact. Note that all events that amount to the execution of protocol actions/messages are recorded as facts in the social

---

[1]Annotations, a form of meta-data, provide data about a program that is not part of the program itself. See `https://docs.oracle.com/javase/tutorial/java/annotations/`

state. In particular, when the social state is updated, the JaCaMo+ artifact provides such information to the focusing JaCaMo+ agents by exploiting proper *observable properties*. Agents are, thus, constantly aligned with the social state.

In JaCaMo+, protocol designers can adopt two different modalities for determining how commitments evolve in their lifecycle: manual or automated. In manual mode, any change in the social state must be explicitly expressed; the social state represents the repository of commitments currently holding, and it changes their status only if a protocol action devises that change. For example, consider the following code excerpt, concerning the action `sendAccept` of Netbill protocol:

```
1  @OPERATION
2  @ROLE( roleName=CUSTOMER_ROLE)
3  public void sendAccept(String item, String price) {
4    RoleId customer = getRoleIdByPlayerName(getOpUserName());
5    RoleId merchant = getRoleIdByGenericRoleName(MERCHANT_ROLE).get(0);
6    try {
7      assertFact(new Fact("acceptedQuotation", item, price, customer.toString()));
8      createCommitment(new Commitment(customer, merchant, "goods", "paid"));
9      ArrayList<Commitment> arrComm = new ArrayList<Commitment>();
10     for (Commitment com : socialState.retrieveCommitmentsByCreditorRoleId(customer))
11       if (com.getConsequent().equals(new Fact("goods"))
12           && com.getDebtor().equals(merchant))
13         arrComm.add(com);
14     for (Commitment com : arrComm) detachCommitment(com);
15   } catch (MissingOperandException e) { ... handle exception ... }
16 }
```

Listing 2. Excerpt of the sendAccept protocol action.

Starting from line 10, the execution of the action may imply a state change for one or more commitments, whose antecedent matches with the asserted fact: those commitments become detached (line 14). This update is not needed in the automated mode: when a fact is asserted, the social state is checked and updated according to the commitment lifecyle, so, for example, a commitment becomes satisfied if its consequent is added. Listing 3 reports the automated version of `sendAccept`. The asserted fact (line 7) will trigger the detachment of those commitments whose antecedent matches with it.

```
1  @OPERATION
2  @ROLE( roleName=CUSTOMER_ROLE)
3  public void sendAccept(String item, String price) {
4    RoleId customer = getRoleIdByPlayerName(getOpUserName());
5    RoleId merchant = getRoleIdByGenericRoleName(MERCHANT_ROLE).get(0);
6    try {
7      assertFact(new Fact("acceptedQuotation", item, price, customer.toString()));
8      createCommitment(new Commitment(customer, merchant, "goods", "paid"));
9    } catch (MissingOperandException e) { ... handle exception ... }
10 }
```

Listing 3. Excerpt of the `sendAccept` protocol action.

Notice that the choice between manual and automated mode does not impact on how agents will be implemented: it is a mechanism aimed at helping protocol designers. To choose automated or manual mode, developers have simply to use the proper constructor of the protocol class, as shown in Listing 4.

```
1  public NetbillProtocol() {
2    super();
3    socialState = new AutomatedSocialState(this);
4  }
```

Listing 4. Automated Social State.

3.2.   Integration of the Interaction Component in Jason

Jason [14] implements in Java, and extends, the agent programming language AgentSpeak(L). Jason agents have a BDI architecture. Each has a belief base, and a plan library. It is possible to specify *achievement* (operator '!') and *test* (operator '?') goals. Each plan has a triggering event (causing its activation), which can be either the addition or the deletion of some belief or goal. The syntax is declarative. In JaCaMo, the beliefs of Jason agents can also change due to operations performed by other agents on the CArtAgO environment, whose consequences are automatically propagated.

In JaCaMo+, Jason agents can enact a commitment-based protocol in the same way they create (or focus on) any other type of artifact. Thus, it is possible to add interaction protocols directly to the MAS specification, or to use those CArtAgO primitives that allow creating and using artifacts. As an example, let us consider a simple multiagent system where agents interact by means of the Netbill protocol. The following *.jcm* definition file specifies a workspace where the interaction protocol is instantiated when the system is deployed, and the involved agents focus on its instance at the beginning of their lifecycle:

```
1  mas netbill {
2    agent customer : customer.asl  {
3      ...
4      focus: nb.netbill
5    }
6    agent merchant : merchant.asl  {
7      ...
8      focus: nb.netbill
9    }
10   ...
11   workspace nb {
12     ...
13     artifact netbill: protocol.NetbillProtocol()
14   }
15 }
```

Listing 5.   The Netbill MAS definition in JaCaMo+.

Here instead an agent plan includes the creation of an interaction artifact and subsequently focuses on it. The string "*netbill-protocol*" is the name of the artifact:

```
1  makeArtifact("netbill-protocol","protocol.NetbillProtocol",[],C);
2  focus(C);
```

In case an agent should focus on an artifact that is created by some other agent, `focusWhenAvailable` will make the agent wait until the artifact becomes available:

```
1  focusWhenAvailable("netbill-protocol");
```

Notice that the proposed techniques are not different from how a Jason agent creates or focuses on any CArtAgO artifact. When an observable event is generated inside an interaction artifact, this notifies it to all agents focusing on it. Observable events amount to the creation or the modification of relevant information inside the social state, e.g. the satisfaction of a commitment or the addition of a social fact. Indeed, all information inside the social state is observable. Also commitments are realized as observable properties. They are represented as terms of the form: $cc(debtor, creditor, antecedent, consequent, status)$, where *debtor* and *creditor* identify the involved agents (or agent roles), while *antecedent* and *consequent* are the commitment conditions. *Status* is the state of the commitment according to the commitment lifecycle. Following the integration of Jason and CArtAgO, observable properties become beliefs for the agent focusing on the artifact. Since social facts are modeled as observable properties, it is

straightforward to program Jason agents that tackle them. To this aim, we extended *Jason* so as to allow the specification of plans, whose triggering events involve commitments. A Jason plan is specified as:

$$triggering\_event : \langle context \rangle \leftarrow \langle body \rangle$$

where *triggering_event* denotes the event the plan handles, the *context* specifies the circumstances when the plan could be used, the *body* is the course of action that should be taken. The *triggering_event* and the *body* can be true or omitted when necessary. In a Jason plan specification, commitments can be used wherever beliefs can be used. Otherwise than beliefs, their assertion/deletion can only occur through the artifact, in consequence to a social state change.

The following template shows a Jason plan triggered when a commitment, that unifies with the one in the plan head, is notified in the agent belief base because it was created in the social state:

$$+cc(debtor, creditor, antecedent, consequent, status) : \langle context \rangle \leftarrow \langle body \rangle$$

The syntax is the standard for Jason plans. *Debtor* and *creditor* are to be substituted by the proper roles. Similar schemas can be used for commitment deletion and for the addition/deletion of social facts. Particularly relevant is the case when the plan allows a debtor of a commitment to take action, after the commitment is detached, in order to comply with its obligation. Also specially relevant is the case when an agent realizes to be creditor of a commitment that was just created and can deliberate whether detaching it.

Commitments can also be used, in Jason plans, inside contexts, as test goals ($?cc(\dots)$), or as achievement goals ($!cc(\dots)$). Addition or deletion of such goals can, as well, be managed by plans, e.g.:

$$+!cc(debtor, creditor, antecedent, consequent, status) : \langle context \rangle \leftarrow \langle body \rangle$$

The plan is triggered when the agent creates an achievement goal concerning a commitment. The agent will, then, act upon the artifact so as to create the desired social relationship. After a successful execution of the plan, the commitment $cc(debtor, creditor, antecedent, consequent, status)$ will hold in the social state, and will be projected onto the belief bases of all agents focusing on the artifact.

We report, as an example, the code of an agent playing the role merchant.

```
1  !start.
2  +!start <- enact("merchant").
3  +enacted(Id,"merchant",Role_Id)
4        <- +enactment_id(Role_Id); ...;
5           !sell.
6  +!sell <- true.
7  +requestedQuote(Item, Merchant)
8        <- ...; sendQuote(Item, Price).
9  +cc(My_Role_Id, Customer_Role_Id, _, "goods","DETACHED")
10        : enactment_id(My_Role_Id)
11        <- sendGoods.
12 +cc(My_Role_Id, Customer_Role_Id, _, "receipt","DETACHED")
13        : enactment_id(My_Role_Id)
14        <- sendReceipt.
```

Listing 6.   The Netbill Merchant agent code in JaCaMo+.

The agent enacts the *merchant* protocol role (operation enact, Listing 6, line 2). The parameter of the action is the name of the role the agent is enacting: if the role does not exist, the action fails. It

is required that the agent keeps a mental note of the ID assigned to its enactment (line 4). After this, the agent activates the goal `sell` that, having an empty body, causes it to simply wait for a customer to start an interaction by asking for a quote[2]. Action `enact` is part of the definition of the abstract class `InteractionProtocol`. After the enactment, the agent is allowed to execute actions that are part of the enacted role. In the example, the merchant agent has a plan whose trigger is a social fact (`requestedQuote`, line 7) representing the fact that a quote request occurred. The agent tackles this situation by sending the requested quote. This is done by executing the protocol action `sendQuote(Item, Price)`, line 8, that modifies the social state by adding the quote at issue realized as observable property (see Listing 1, line 11). In turn, the artifact will notify to the customer the quote by propagating the new observable property into its belief base. The plans at lines 9 and 12 are typical examples of how an agent can tackle the progression of commitments along their lifecycle. The first plan handles the satisfaction of the commitment regarding goods dispatch (`cc(Customer_Role_Id, My_Role_Id, _, "goods","DETACHED")`), for which the merchant was creditor. Since the commitment is detached, the merchant is now expected to bring about the consequent condition. The plan simply sends the agreed item (action `sendGoods`). The second plan again tackles the detachment of a commitment concerning the merchant as debtor: the merchant uses the action `sendReceipt` to fulfill it.

In order to understand the interaction of a customer with a merchant, we report also the Jason program of a possible customer. The customer asks for a quote right after enacting the role (through buy). The program includes a plan (see Listing 7, line 7) that is activated when the commitment `cc(My_Role_Id, Customer_Role_Id, AcceptedQuotation, "goods","CONDITIONAL")` is notified to the customer by the Netbill interaction artifact. Such a commitment means that the merchant promises to send the goods in case the customer accepts the quote. It is created by the merchant as a social consequence of the protocol action `sendQuote` (see Listing 1, line 7). The customer's plan in Listing 7 is very simple: the agent accepts the quote by executing `sendAccept`[3], which creates the conditional commitment `cc(customer, merchant, "goods", "paid")`, and detaches the merchant's commitment tackled at line 9 of the merchant's program (Listing 6). This event is notified to the merchant by the artifact, so the merchant's plan aimed at sending the goods is activated. Thus, the commitment of the customer to pay becomes detached and the customer is compelled to pay.

```
1  !start.
2  +!start <-  enact("customer").
3  +enacted(Id,"customer",My_Role_Id)
4         <- +enactment_id(My_Role_Id); ...;
5             !buy.
6  +!buy <- sendRequest("item").
7  +cc(Merchant_Role_Id, My_Role_Id, AcceptedQuotation, "goods","CONDITIONAL")
8         :   enactment_id(My_Role_Id)
9             & sendQuote(Item,Price,Merchant_Role_Id)
10            & not goods(Item)
11        <- ...; sendAccept(Item, Price).
12 +cc(My_Role_Id, Merchant_Role_Id, _, "paid","DETACHED")
13        :   enactment_id(My_Role_Id)
14        <- sendEPO.
15 +cc(Merchant_Role_Id, My_Role_Id, AcceptedQuotation, "goods", "SATISFIED")
16        :   enactment_id(My_Role_Id)
17            & sendQuote(Item,Price,Merchant_Role_Id)
18            & not acceptedQuotation(_,_)
19        <- ...; sendAccept(Item, Price).
```

---

[2]The reason for relying on the explicit goal `sell` will become clear in Section 5.

[3]The code is not reported in the paper but it is available in the code repository.

```
20  +goods ( Item )
21          :   enactment_id (My_Role_Id)
22              & not acceptedQuotation ( Item , Price )
23          <-  sendRequest ( Item ).
```
Listing 7.   The Netbill Customer agent code in JaCaMo+.

The program accommodates also alternative types of interaction where the customer, instead of taking the initiative, waits for some interesting offer. We discuss this case in Section 5.

4.   The Gold Miners Scenario

The Gold Miners scenario was initially proposed at CLIMA VII [4] to challenge researchers in the multi-agent community; since then, it has been used as a reference example in many other works [13, 40]. It consists in developing a multi-agent system to solve a cooperative task in a dynamically changing environment. The environment is a grid-like world where agents can move from one cell to a neighboring cell if it contains no agent or obstacle. Gold nuggets can appear in the cells at any time. Agents, operating as a team, are expected to explore the environment, avoid obstacles, and collect as much gold as possible to be dropped into a depot. Each agent can carry one gold nugget at a time (an agent that is not carrying gold is *free*). Agents have only a local view on their environment because they can only see pieces of gold in the adjacent cells; however, they can communicate with the other agents for sharing their findings.

The solution we present in this section[5] exploits the basic infrastructure used in the JaCaMo tutorial[6]: in particular, a world simulator is at the basis of the architecture; the simulator keeps a complete state of all the agents, nuggets, and obstacles. Each agent, however, has just a limited view of the events occurring inside the simulator; this is due to the fact that the agents' perceptions are simulated by means of standard CArtAgO artifacts that, for each agent, capture only a specific subset of the events generated by the simulator. Upon this underlying architecture we implement our JaCaMo+ solution.

In JaCaMo+, a solution to the Gold Miners consists of two elements: the Mining Protocol (i.e., the interaction artifact), and the Miner (i.e., the agents). The Mining Protocol is shown in Table 2. The pro-

Table 2.   Mining Protocol: actions and their social meaning; $m_1, \ldots, m_n$ are the miners in the team.

$volunteer(m_i, X, Y, Dist)$:	$\{\text{create}(C(m_i, m_j, \bigwedge_{m_k \neq i} agree(m_k, X, Y), pick(X, Y) \cdot drop(X, Y))) \mid m_{j \neq i}\}$
$offer(m_i, X, Y, Dist)$:	$\{\text{create}(C(m_i, m_j, \bigwedge_{m_k \neq i} agree(m_k, X, Y), pick(X, Y) \cdot drop(X, Y))) \mid m_{j \neq i}\}$
$withdraw(X, Y, m_i)$:	$\{\text{cancel}(C(m_i, m_j, \_, pick(X, Y) \cdot drop(X, Y))) \mid m_{j \neq i}\}$
$agree(m_i, X, Y)$:	*commitment progression*
$pick(X, Y)$:	*commitment progression*
$drop(X, Y)$:	*commitment progression*

tocol involves only one role, *miner*, because the scenario assumes a team of homogeneous agents. Action *volunteer* allows agent m_i to offer to go and pick up a gold nugget, that the agent m_i itself perceived at coordinates X, Y, and whose distance from m_i is $Dist$, and then to drop the nugget at the depot. If m_i was the first agent to perceive that piece of gold, a set of commitments is created in the social state as

---

[4] http://www.staff.science.uu.nl/~dasta101/publication/CLIMA07contest.pdf

[5] The code is available at http://di.unito.it/2COMM.

[6] http://jacamo.sourceforge.net/tutorial/gold-miners/

an effect. Each of these commitments has the form $C(m_i, m_j, \bigwedge_{m_k \neq i} agree(m_k, X, Y), pick(X, Y) \cdot drop(X, Y))$. It is directed towards one of the other agents m_j, and binds m_i to the temporal expression $pick(X, Y) \cdot drop(X, Y)$ in case all other agents agree that m_i performs the task. Action *offer* is executed when an agent becomes aware that a new piece of gold was found through the commitments created by the miner that found the gold. The action aims at creating a counter-offer and achieves this purpose by creating the same kind of commitments created by *volunteer*. By executing action *withdraw*, the debtor agent cancels a commitment previously created. Actions *agree*, *pick*, *drop* cause commitment progression.

In principle, each agent enacting the *miner* role could implement an internal strategy which is independent of the strategies used by its teammates; the only constraint is that all the miners use the actions made available by the Mining Protocol to interact. The goal of this example is to show how the strategy proposed in [13, 40] for Jason agents can be realized in JaCaMo+. For this reason, we assume that all the mining agents adopt the same strategy. Each miner has two mutually exclusive goals: "*look for gold*" and "*drop gold*". At the beginning of the simulation, all the miners focus on the same instance of the Mining Protocol, previously created by the system. Moreover, they are randomly placed on the map, which is managed by an instance of the artifact MiningPlanet. All the interacting agents will be notified of changes occurred to the observable properties of such artifacts. Each agent initially has the active goal "look for gold". The search for gold is carried out by moving towards a randomly selected target position. The hope is that, while moving, the miner will walk by a piece of gold, or it will be notified by another agent that gold has been found in a place nearby. In particular, when an agent gets close to a location that contains a piece of gold, an event `+cell(X,Y, gold)` occurs in the artifact MiningPlanet. The event represents that the agent perceived the presence of gold in the cell of coordinates X and Y.

When a new piece of gold is identified by a miner, three alternative behaviors are possible:

- (*R1*) The miner is not carrying gold, nor it is committed to pick up gold: the miner volunteers to pick it up and drop it at the depot (Listing 8, lines 2–6).

- (*R2*) The miner is not carrying gold, but it is already committed to pickup some more distant nugget: it withdraws, canceling the previous commitment to pick and drop the old nugget, and volunteers to pick up the new one and drop it at the depot (lines 8–18). (*R2'*) When a miner cancels a detached commitment, all the others are made aware of this by the social event represented by the violation of that commitment. In this cooperative environment, however, the violation of a commitment does not entail a sanction to the responsible miner but, rather, the violation denotes that a gold nugget is now available to be picked up. Hence, this cancellation will cause other miners to offer to manage the piece of gold, that was left unassigned after withdrawal (lines 54–62).

- (*R3*) The miner is already carrying a gold to the depot, it cannot handle another gold, so it volunteers using as $Dist$ a conventional high value, in order to stimulate the other miners to make their offers, without possibly being chosen (lines 20–26). (*R3'*) In the special case in which no other miner makes a counter-offer (all are busy), and instead they agree to leave the new piece of gold to the agent that found it, this agent executes withdraw of the, now detached, commitment to pick and drop the new nugget (lines 64–69).

When a miner volunteers or makes an offer, commitments are created in the social state. All the other miners will tackle this eventuality in the following way:

- (*R4*) if the miner is already committed to take another piece of gold, it executes *agree* (lines 28–33).

- (*R5*) if the miner offered to take the same piece of gold but its distance from gold is greater than that of its teammate, the miner, which has a cooperative behavior, withdraws and performs *agree* (Listing 8, lines 35–41).

- (*R6*) if the miner is not carrying gold, nor it is committed to pick up gold: it decides whether to offer or not (in the latter case it will execute *agree*). A miner offers to take gold only if its distance to the gold location is less than the distance of the agent that has found the gold (lines 43–52).

The conditional commitment, that is created by volunteering (offering), is the one that, once detached, will activate the plan by which the agent will handle the gold nugget. So, finally, when a miner volunteered or offered to pick and drop a piece of gold, and all the others agreed to this (meaning that the corresponding commitment is detached), the miner starts the plan to handle that piece of gold ((*R7*), line 71–76 and (*R7'*), 78–80). Indeed, when an agents creates a conditional commitment, it must be ready to bring about the consequent of such a commitment whenever the antecedent will become true.

```
 1 // Plan implementing (R1)
 2 +cell(X,Y,gold) : enactment_id(My_Role_Id) & pos(MyX, MyY)
 3         & not cc(My_Role_Id, _, _, _, "CONDITIONAL") &
 4         & not cc(My_Role_Id, _, _, _, "DETACHED")
 5         <- jia.dist(X, Y, MyX, MyY, Dist);
 6            volunteer(X, Y, Dist).
 7 // Plan implementing (R2)
 8 +cell(X,Y,gold) : enactment_id(My_Role_Id) & pos(MyX, MyY)
 9         & cc(My_Role_Id, _, Agree, PickThenDrop, STATUS)
10         & (STATUS == "CONDITIONAL" | STATUS == "DETACHED")
11         & not carrying(OldX, OldY)
12         & .term2string(PickThenDrop, T) &
13         & jia.getCoord1(T, OldX) & jia.getCoord2(T, OldY) & X \== OldX & Y \== OldY
14         <- .drop_intention(handle(gold(OldX, OldY))); -handling_gold(OldX, OldY);
15            jia.dist(X, Y, MyX, MyY, Dist);
16            withdraw(OldX, OldY);
17            -detach;
18            volunteer(X, Y, Dist).
19 // Plan implementing (R3)
20 +cell(X,Y,gold) : enactment_id(My_Role_Id)
21         & cc(My_Role_Id, _, _, Drop, "DETACHED")
22         & .term2string(Drop, T)
23         & jia.getCoord1(T, OldX) & jia.getCoord2(T, OldY)
24         & not offer(X, Y, _, My_Role_Id)
25         & pick(OldX, OldY)
26         <- volunteer(X, Y, 1000).
27 // Plan implementing (R4)
28 +cc(Other_Role_Id, My_Role_Id, _, PickThenDrop, "CONDITIONAL") : enactment_id(My_Role_Id)
29         & .term2string(PickThenDrop, T)
30         & jia.getCoord1(T, X) & jia.getCoord2(T, Y)
31         & cc(My_Role_Id, _, _, _, STATUS)
32         & (STATUS == "CONDITIONAL" | STATUS == "DETACHED")
33         <- agree(X, Y).
34 // Plan implementing (R5)
35 +cc(Other_Role_Id, My_Role_Id, _, PickThenDrop, "CONDITIONAL") : enactment_id(My_Role_Id)
36         & .term2string(PickThenDrop, T)
37         & jia.getCoord1(T, X) & jia.getCoord2(T, Y)
38         & offer(X, Y, Dist, My_Role_Id) & offer(X, Y, Other_Dist, Other_Role_Id)
39         & (Dist > Other_Dist)
40         <- agree(X, Y);
41            withdraw(X, Y).
```

```
42  // Plan implementing (R6)
43  +cc(Other_Role_Id, My_Role_Id, _, PickThenDrop, "CONDITIONAL") : enactment_id(My_Role_Id)
44          &   pos(MyX, MyY)
45          &   .term2string(PickThenDrop, T)
46          &   jia.getCoord1(T, X) & jia.getCoord2(T, Y)
47          &   not cc(My_Role_Id, _, _, _, "DETACHED")
48          &   not cc(My_Role_Id, _, _, _, "CONDITIONAL")
49          &   offer(X, Y, Other_Dist, Other_Role_Id)
50          <-  jia.dist(X, Y, MyX, MyY, Dist);
51              if (Dist < Other_Dist) { offer(X, Y, Dist); }
52              else { agree(X, Y); }.
53  // Plan implementing (R2')
54  +cc(Other_Role_Id, My_Role_Id, _, PickThenDrop, STATUS) : enactment_id(My_Role_Id)
55          &   pos(MyX, MyY)
56          &   (STATUS == "TERMINATED" | STATUS == "VIOLATED")
57          &   not cc(My_Role_Id, _, _, _, "DETACHED")
58          &   not cc(My_Role_Id, _, _, _, "CONDITIONAL")
59          &   offer(X, Y, Other_Dist, Other_Role_Id)
60          &   not offer(X, Y, _, My_Role_Id)
61          <-  jia.dist(MyX, MyY, X, Y, Dist);
62              offer(X, Y, Dist).
63  // Plan implementing (R3')
64  +cc(My_Role_Id, Other_Role_Id, _, PickThenDrop, "DETACHED") : enactment_id(My_Role_Id)
65          &   .term2string(PickDotDrop, T)
66          &   jia.getCoord1(T, X) & jia.getCoord2(T, Y)
67          &   cc(My_Role_Id, _, _, Other_PickThenDrop, "DETACHED")
68          &   PickThenDrop \== Other_PickThenDrop
69          <-  withdraw(X, Y).
70  // Plan implementing (R7)
71  +cc(My_Role_Id, _, _, _, "DETACHED") : enactment_id(My_Role_Id)
72          &   .term2string(PickDotDrop, T)
73          &   jia.getCoord1(T, X) & jia.getCoord2(T, Y)
74          &   not detach
75          <-  -free; +detach;
76              !init_handle(gold(X,Y)).
77  // Plan implementing (R7')
78  +!init_handle(gold(X,Y)) : ...
79          <- ... pick(X, Y); ...
80              ... drop(X, Y); ... .
```

Listing 8.   The Gold Miner agent code in JaCaMo+.

Listing 9 outlines the JaCaMo+ Mining Protocol implemented as an interaction artifact.

```java
1   public class MiningProtocol extends ProtocolArtifact {
2     public static String ARTIFACT_TYPE = "MiningProtocol";
3     public static String MINER_ROLE = "miner";
4     private int numberOfMiners = MINER_NUMBER;
5     static { addEnabledRole(MINER_ROLE, Miner.class); }
6     ...
7     public MiningProtocol() {
8       super();
9       socialState = new AutomatedSocialState(this);
10    }
11    @OPERATION   public void volunteer(int x, int y, int dist) { ... }
12    @OPERATION   public void offer(int x, int y, int dist) {
13      try {
14        CompositeExpression cons = null;
15        RoleId offeringMiner = getRoleIdByPlayerName(getOpUserName());
16        RoleId groupMiner = new RoleId(MINER_ROLE);
17        ArrayList<RoleId> enactedMiners = getRoleIdByGenericRoleName(MINER_ROLE);
18        LogicalExpression antec = new Fact("true");
19        for (int i = 0; i<numberOfMiners-TEMP_LIMIT; i++) {
```

```
20            if  (!( enactedMiners . get ( i ). equals ( offeringMiner ))) {
21            CompositeExpression acc =
22                new CompositeExpression ( LogicalOperatorType .AND,
23                new Fact ("agree", x, y, enactedMiners . get ( i ). toString ()), antec );
24            antec = acc ;
25            }
26        }
27        cons = new CompositeExpression ( LogicalOperatorType .THEN,
28            new Fact ("pick", x, y), new Fact ("drop", x, y ));
29        Commitment groupCommit =
30            new Commitment ( offeringMiner , groupMiner , antec , cons );
31        createAllCommitments ( groupCommit );
32            assertFact (new Fact ("offer", x, y, dist , offeringMiner . toString ()));
33    } catch ( MissingOperandException e) { ... handle exception ... }
34    }
35    @OPERATION  public void withdraw (int x, int y) {
36       try {
37        RoleId askingMiner = getRoleIdByPlayerName ( getOpUserName ());
38        CompositeExpression cons ;
39        Fact f;
40        List <Commitment> comms =
41            socialState . retrieveCommitmentsByDebtorRoleId ( askingMiner );
42        if (comms != null) {
43         for (Commitment c : comms) {
44            cons = ( CompositeExpression )(c. getConsequent ());
45            f = (Fact )( cons . getLeft ());
46            if (( int )( f. getArguments ()[0]) == x &&
47                ( int )( f. getArguments ()[1]) == y)
48            cancelCommitment (c );
49        }
50       }
51    } catch ( MissingOperandException e) { ... handle exception ... }
52    }
53    @OPERATION  public void agree (int x, int y) { ... }
54    @OPERATION  public void pick (int x, int y) {
55       try {
56        RoleId pickingMiner = getRoleIdByPlayerName ( getOpUserName ());
57        Fact f = new Fact ("pick",x,y);
58        assertFact (f );
59    } catch ( MissingOperandException e) { ... handle exception ... }
60    }
61    @OPERATION  public void drop (int x, int y) { ... }
62    public class Miner extends PARole {
63       public Miner( String playerName , IPlayer player ) {
64        super (MINER_ROLE, player );
65       }
66    }
67 }
```

Listing 9. The Gold Miner artifact in JaCaMo+.

The `MiningProtocol` class extends the `ProtocolArtifact` class of JaCaMo+. It includes three class variables, among which `numberOfMiners` stores the number of miners cooperating in the team. According to the original GoldMiners competition, such a number is predetermined and does not change at runtime. In our implementation the number of miners is set by the configuration constant `MINER_NUMBER`. The class constructor sets the field `socialState`, which is inherited from `ProtocolArtifact`, to a new instance of `AutomatedSocialState()`. This means that commitment states progress automatically according to the events (i.e., operations) occurring inside the artifact, relieving the programmer to explicitly program this part. The rest of the class specifies all the protocol operations. Each protocol action is re-

alized as a method tagged with the `@OPERATION` annotation. For the sake of simplicity, the excerpt reported in the Listing details only part of such operations. Operation `offer` is executed by a miner, after it became aware that a new piece of gold was found through the commitments created by the miner that found it, and after it decided to create a counter-offer because it is closer to that gold nugget. The execution of `offer` creates a set of commitments, towards all other miners (lines 29–31). When a miner decides to cancel a set of commitments, it executes `withdraw`. Commitment cancellation is performed at line 48. Finally, when a miner executes `pick`, a fact is created in the social state to record the event (line 58).

5. Discussion

To show the added value of the interaction artifacts of JaCaMo+, we compare them with two alternative approaches to interactions: message-based protocols, as the one used in [13] for solving the Gold Miners in JaCaMo, and the interaction components in [70].

The first observation is that interaction artifacts enable a form of flexibility that neither message-based protocols, nor interaction components can obtain. These two approaches are basically prescriptive: they describe a specific sequence of steps through which an interaction can correctly evolve. In real world cases, however, there exist alternative paths that bring about the same result. Prescriptive approaches either ignore these alternatives, or encode them by making the resulting protocol (or component) awkward to deal with. Interaction artifacts, on the contrary, define protocols based on commitments, without imposing any strict ordering of the messages (or of the actions) besides those imposed by the commitment conditions. It follows that agents generally have greater flexibility in deciding which protocol action to execute. For instance, let us consider the Netbill protocol in Table 1. The most natural way to interpret the interaction is that the customer starts it by asking a merchant the price of a given item (*sendQuote(Item, Price)*). Let us consider Listing 6, in Section 3, reporting the code of the merchant. In this code, at line 6, the plan `+!sell <- true` realizes this reactive behavior: the merchant waits for quote requests. However, this is just one of the possible evolutions of the interaction between a merchant and a customer. Even a merchant can take initiative and start an interaction: it can make an offer to the customer by sending a quotation for an item, despite the customer did not request it. In the real world, this is, for instance, the case of special discounts that sometimes are advertised by vendors. It is easily possible to add also this behavior to the merchant agent, without disrupting its possibility to interact with other agents through the same protocol. In order to realize this behavior, it is sufficient to substitute the plan at line 6 with the plan `+!sell <- sendQuote("item", "price")`. The merchant starts by sending a quote. The customer autonomously assesses whether such an unexpected offer is valuable, and in the positive case takes advantage of it. The merchant can also start an interaction by sending goods directly to the customer. The rest of the program is the same as the one reported in Listing 6 and also the artifact that allows the interaction does not change. This is the case of an agent offering products directly at the customer's place. This behavior is obtained by using, instead, the plan `+!sell <- sendGoods("item")`. Again, the customer assesses whether the goods are of any interest; if so, it will ask a quotation by performing `sendRequest(Item)` with `Item` unifying `goods`, and then decide whether completing the purchase or not. For what concerns the customer, in order to be able to interact with the two just described kinds of merchant, it is sufficient to substitute the plan at line 6, in Listing 7 with `+!buy <- true` so that the customer leaves the initiative to the merchant. It is evident that all these

scenarios (and programs) bring to a situation where both merchant and customer have satisfied their own goals. They amount to alternative interactions that can be legally (according to the protocol) undertaken by the two interacting agents, alternative interactions that the use of commitments allows. This is due to the fact that commitments, which avoid imposing orderings over the agents' actions, respect the principle of Minimal Critical Specification [18]. Quote: "..., it is a mistake to specify more than is needed because by doing so options are closed that could be kept open. This premature closing of options is a pervasive fault in design.". This is an important difference with respect to automata-based approaches, which are usually overly prescriptive. Also in the Gold Miners scenario it is possible to implement miner agents, that encode alternative strategies, without modifying the protocol (hence the interaction artifact). For instance, it would be possible to realize selfish agents which do not share the position of newly found pieces of gold when they are already carrying one to the depot, and keep the finding for themselves, for later use. It would be enough to remove rule (R3) and the plan which implements it.

The JaCaMo+ proposal, thus, enjoys a form of decoupling between the design of the agents and the design of the interaction that the other approaches lack of. Let us briefly compare the JaCaMo implementation of Dijkstra's Dining Philosophers [52] with an implementation in JaCaMo+ (see [6] for details). The JaCaMo solution involves two roles: *waiter* and *philosopher*. The former is played by an agent initializing the artifact (i.e., the table) used for the coordination; the latter is the role played by the agents to be coordinated. An agent *philosopher* follows the general loop of thinking and eating. To eat, however, the philosopher has to acquire two resources (i.e., the forks), shared with others. The acquisition of the forks could lead the philosophers into a deadlock condition; therefore, this step is coordinated by the artifact. The following two Jason plans give an intuition of how the deadlock is avoided.

```
1 +!acquireRes: my_left_fork(Left) & my_right_fork(Right)
2         <- in("ticket");
3            in("fork",Left); in("fork",Right).
4 +!releaseRes: my_left_fork(Left) & my_right_fork(Right)
5         <- out("fork",Left); out("fork",Right);
6            out("ticket").
```
Listing 10.   An excerpt of the philosopher in JaCaMo [52].

The plan starting at line 1, Listing 10, handles the request of the forks from a philosopher. Notice that before obtaining the forks, a philosopher needs to get a `ticket` (line 2) by means of the (tuple space) `in` operation; this operation guarantees that the agent asks for two forks in mutual exclusion. Once obtained the ticket the agent can acquire the two forks without the risk of getting stuck in a deadlock condition. The plan starting in line 4, on the other hand, handles the release of the two forks. Also this step involves an operation of the underlying artifact, `out`, which corresponds to the return of the resources to the artifact. Notably, the agent has to release the `ticket`, too; see line 6. This is essential for enabling other philosophers waiting for the forks. These two plans show a potential vulnerability of the solution: the programmer is in charge of using the two primitives `in` and `out` correctly (i.e., in the proper order); but she is not supported in her job since the meaning of the two primitives is hidden inside the artifact.

In JaCaMo+, instead, a philosopher who decides to eat executes an operation (provided by the Dining Philosophers protocol artifact), which creates a commitment $cc($`phil`$, philosopher, available(Left, Right), return(Left, Right))$. The intuitive meaning is that `phil` commits with all the other philosophers that in case it gets the forks, it will then return them. The JaCaMo+ philosopher agent program must, then, contain a plan to handle the situation in which the above commitment becomes detached because `phil` obtained the forks. The plan will try to achieve the consequent condition, i.e. to return the forks. A possible plan is the following one:

```
1 +cc(My_Role_Id, "philosopher", available(Left,Right), returnForks(Left,Right),"DETACHED")
2        : enactment_id(My_Role_Id) & my_left_fork(Left) &
3          my_right_fork(Right)
4     <- !eat(Left, Right);
5          returnForks(Left, Right).
```

Listing 11.   A plan in JaCaMo+ to accomplish a `philosopher`'s goal and commitment.

Observe that the triggering event (Listing 11, line 1) is the change of state of the commitment in which the current agent appears as debtor. Since the commitment is now detached, the agent has to satisfy it by returning the forks. Indeed, the context (lines 2-3), makes clear that the agent now holds two forks (*Left* and *Right*). More interestingly, the body of the plan (lines 4-5) has two steps: first, the agent accomplishes its individual goal of eating by `!eat(Left, Right)`, and, then, the agent will satisfy the commitment by returning the two forks, and thus accomplishing the necessary coordination. The advantage of using JaCaMo+ protocol artifacts should now be evident. The programmer does not need to know the internal logic with which the artifact resolves conflicts, nor she/he needs to know and use structures that are internal to the artifact as, instead, happens in the solution proposed by Ricci et al. [52], where the programmer has to use a `ticket` to guarantee the mutual exclusive access to the forks. The proper usage of the ticket, however, is left to the ability of the programmer, who could forget to return the ticket after using the forks. In JaCaMo+, instead, the programmer is driven by the commitments in which the agent under development will be involved. The act of returning the forks, for example, is not a procedure that the programmer has to remember to do, but becomes a new goal that the programmer has to fulfill in order to satisfy the agent's commitments, and hence to satisfy the agent's interactions. In other words, JaCaMo+ shifts the focus from programming reactions to signals, to programming plans for achieving conditions that satisfy both the agent's goals and the agent's commitments.

More generally, JaCaMo+ implicitly suggests an approach to agent programming driven by goals and commitments. Telang et al. [61] have shown how goals and commitments are profoundly related with one another. Leveraging on this relation, it is possible to devise code templates as basic building blocks for the agent body. JaCaMo+ supports this approach. To make this point clearer, let us consider now the Gold Miners solution in JaCaMo+ (Section 4), and compare it with the one proposed in [13, 40][7]. JaCaMo+ commitment-based protocols exploit an explicit causal relation between two events: the creation of a conditional commitment, and the detachment of that very commitment. For instance, action *volunteer* creates conditional commitments of form $C(m_i, m_j, \bigwedge_{m_{k \neq i}} agree(X, Y, m_k), pick(X, Y) \cdot drop(X, Y))$. Such commitments can be seen as a sort of norms that, once asserted in the social state, bind the debtor to bring about their consequent in case the antecedent occurs. The same commitment, however, can be seen as a piece of data existing in the social state that, differently from other forms of data, has a state and such a state evolves in a precise manner, that is specified by the commitment lifecycle. Relying on these two observations, a programmer has just to focus on the set of commitments in which the agent will be involved and, in particular, the programmer will pay particular attention to the *state changes* occurring in these commitments. For instance, since the action *volunteer* is invoked in the miner's program (Listing 9), and since the programmer knows that such action yields the creation of commitments, the programmer will also know that it is necessary to include plans for tackling the state changes of such commitments that are relevant for the agent itself. It will, then, include a plan whose

---

[7]For the sake of completeness, at http://jacamo.sourceforge.net/tutorial/gold-miners/ it is possible to find a JaCaMo implementation of the Gold Miners but artifacts are not used for realizing interaction. They are used only to realize the individual agent views of the environment in which miners search for gold.

triggering event is the *detachment* of one such commitment (event after which the miner is expected to pick up the gold nugget and drop it at the depot), and whose body brings about the commitment consequent condition (Listing 8, the plans implementing (R3') and (R7)). It will also include a plan for tackling the *violation* of a similar commitment by another agent, in order to enable a reorganization of task assignments (plan implementing (R2')), as well as plans for tackling commitment *creation*, see (R4), (R5), and (R6). This characteristic is also the reason why it was possible to realize the three variants of Netbill merchant by just changing one line of code. The action `sendQuote` creates two commitments, whose detachments are handled by the remaining part of the code. It does not matter at which stage of the interaction detachments occur, the agent will tackle them appropriately. First steps towards the realization of an agent programming methodology, that is based on these premises, are described in [6].

In the proposal in [13, 40], instead, agents interact by message exchange. There are two types of agents: the *miners* and a special *leader* agent that coordinates the miners. Let us consider the *gold allocation protocol* used in [40], by which a leader assigns the task of picking up a piece of gold to one of the miners. In such a protocol, all miners bid for a new piece of gold by sending a message *bid*. The leader, once it has received all the bids, answers to the winning miner with the message *allocatedTo(gold(X,Y))*. The relationship between *allocatedTo(gold(X,Y))* and *bid*, however, is not evident to the programmer, who, in order to understand that it is necessary to add a plan for tackling such an event, has no other resource but analysing the protocol. Similar considerations can be drawn when comparing JaCaMo+ with the interaction component proposed in [70] that, as already noted, is thought of as an automaton encoding and orchestrating interaction in a prescriptive way. Since the causal relations between what an agent does and the obligations it generates are hidden inside the interaction component, a programmer cannot but program the agent so that it reacts to such obligations. This situation is very similar to programming agents that react to incoming messages.

6.   Conclusions

In this paper we have presented the JaCaMo+ framework that integrates commitment-based interaction inside JaCaMo. As such, JaCaMo+ realizes the vision of MERCURIO framework [10, 2] and complements the proposal in [4], where an approach to Socio-Technical Systems, based on the Jade framework, is presented.

The modular nature of the implementation facilitates the development of extensions for tackling richer contexts. In particular, as [3] puts forward, most studies in the research area on multiagent systems are focused only on features of agents, while those that support the need of representing the environment mostly disregard the plurality of data, relying on propositional representations: for going beyond the propositional case, it is necessary to rely on an information system (data awareness). Although data-awareness is not yet realized in multiagent systems, the literature contains some independent efforts, among which [19, 45, 21, 23, 3], that tackle aspects of this direction of research. Among these, [21] provides an information-centric representation of commitments that distinguishes between schemas and their instances, and relies on relational database queries. DACMAS [45] concerns commitment-based MASs in a data-aware context, and shows that when a DACMAS is state-bounded, i.e., the number of data that are simultaneously present at each moment in time is bounded, verification of rich temporal properties becomes decidable. Data-awareness is definitely a facet that we mean to introduce in JaCaMo+ in order to enable the realization of systems that are suitable to tackle the complexity of reality.

We are also interested in tackling, in the implementation, a more sophisticate notion of social context and of enactment of a protocol in a social context [8], as well as to introduce a typing system along the line of [5]. The modularity also enables a fully fledged range of verifications and helps modularizing the verification of properties inside a multiagent system, thus enhancing the correctness quality. In particular, it becomes possible to perform the analysis of properties at the level of protocol specification rather than on the system as a whole. The outcome will hold for any instance of the artifact that will be created. Of course, for each use it will be necessary to check that the usage of the artifact, done by a specific agent, conforms to the specification but this is a much simpler kind of verification [7]. Among the kinds of verification that can be performed, modularity enables lightweight forms of verification like type checking [54, 25, 5, 4].

The specification and control of interaction is relevant for areas like the organizational theory and electronic institutions where the focus is generally orthogonal to the one posed on interaction protocols, as it concerns the modeling of the structure rather than of the interaction [2]. Intuitively, an organization establishes a society of agents, that is characterized by a set of organizational goals and a set of norms. Agents, playing one or more roles, should accomplish the societal goals respecting the norms. Electronic institutions [30, 24], similarly to organizations, use norms for regulating the agents' interactions. Differently from them, however, they have no goals of their own, and can be considered as a sort of monitors controlling the agents' behaviors. For example, the abstract architecture of e-institutions envisioned by Ameli [34, 1] places a middleware, made of governors and staff agents, between participating agents and an agent communication infrastructure. The environment is nothing agents can sense and act upon but rather it is a conceptual one. Agents communicate with each other by means of speech acts and, behind the scene, the middleware mediates such communication thanks to a body of norms and laws. Organizations add to the picture the notion of organizational goal, distributing it through the roles that make up the organization [41]. Among the current proposals, the organizational infrastructure in [39] as well as JaCaMo is based on $Moise^+$, which allows both for the enforcement and the regimentation of the rules of the organization. This is done by defining a set of conditions to be achieved and the roles that are permitted or obliged to perform them.

In the solution we proposed, interaction artifacts are orthogonal to the organizational dimension of JaCaMo provided by $Moise^+$, in agreement with the MERCURIO vision. Nevertheless, a JaCaMo+ interaction protocol can itself be seen as a kind of organization as it gives structure to a multiagent system. The implementation sees to create a concrete environment around a protocol in use, that agents manipulate by means of the actions provided by the protocol itself. In doing so agents create commitments, that put them in relationship, that start interactions, that have a normative value, and that progress along with the agent actions. As future work, it would be interesting to harmonize the commitment-based approach to interaction with the obligation-based approach to interaction on a uniform basis, in order to make the framework suitable to tackle a wider range of possible applications. From an organizational perspective, the advantages would be numerous. The multiagent system would be more flexible and, as such, capable to take advantage from unexpected opportunities, to adapt to exceptional conditions in the environment, and to support applications where a number of partners need to interact and coordinate their activities even though they are not members of a recognized organization. This is the standard, for instance, in the constructions area and in cross-organizational settings.

Commitment violations are a kind of commitment state change that a JaCaMo+ interaction artifact can detect and notify to all the agents focusing on it. The decision of how to use such information is currently left to the agent programmers, who will decide whether and how managing the possible viola-

tions. An interesting future development would be to provide the means for harmonizing the reactions produced by the single agents in the MAS, letting them be driven by the norms of the system, when this is appropriate for the application at issue. The idea is to combine JaCaMo+ with proposals from the area of e-institutions. In this area, usually agents are indirectly controlled via a set of norms, regulating the ways in which agents can or cannot behave in the system. All agents have to respect the norms lest being sanctioned by the institution. We would like to come to a vision that integrates violation management inside commitment-based interactions in a manner that makes the involved agents answer to violations in ways that comply with the norms of the system. We deem as particularly interesting, in this respect, the OCeAN meta-model for artificial institutions [36], because it includes a notion of commitment. A possible architecture for OCeAN is discussed in [48].

Our proposal is also backed up by the practical rules discussed in [61], which highlight how goals and commitments are each other related. A first implication is that it is possible to devise methodologies for programming the Jason agents. A second implication concerns self-* applications. Since the agent's autonomy is not constrained, agents maintain the ability of autonomously taking advantage from opportunities and of properly reacting to unexpected events (self-adaptation). For instance, by finding a way for accomplishing an organizational goal taking into account the current state of the MAS, which is hardly foreseeable at design time. Moreover, the interplay between goals and commitments opens the way to the integration of self-governance mechanisms into organizational contexts.

Acknowledgements.

References

[1] Arcos, J. L., Noriega, P., Rodríguez-Aguilar, J. A., Sierra, C.: E4MAS Through Electronic Institutions, *Environments for Multi-Agent Systems III, Third International Workshop, E4MAS 2006, Hakodate, Japan, May 8, 2006, Selected Revised and Invited Papers* (D. Weyns, H. V. D. Parunak, F. Michel, Eds.), number 4389 in Lecture Notes in Computer Science, Springer, 2006.

[2] Baldoni, M., Baroglio, C., Bergenti, F., Boccalatte, A., Marengo, E., Martelli, M., Mascardi, V., Padovani, L., Patti, V., Ricci, A., Rossi, G., Santi, A.: MERCURIO: An Interaction-oriented Framework for Designing, Verifying and Programming Multi-Agent Systems, *Proceedings of The Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2010), Lyon, France, August 30 - September 2, 2010* (O. Boissier, A. E. Fallah-Seghrouchni, S. Hassas, N. Maudet, Eds.), number 627 in CEUR Workshop Proceedings, CEUR-WS.org, 2010.

[3] Baldoni, M., Baroglio, C., Calvanese, D., Micalizio, R., Montali, M.: Towards Data- and Norm-aware Multiagent Systems, *Post-Proc. of the 4th International Workshop on Engineering Multi-Agent Systems, EMAS 2016, Revised Selected and Invited Papers* (M. Baldoni, J. P. Müller, I. Nunes, R. Zalila-Wenkstern, Eds.), number 10093 in LNAI, Springer, 2016, ISBN 978-3-319-50982-2, ISSN 0302-9743.

[4] Baldoni, M., Baroglio, C., Capuzzimati, F.: A Commitment-based Infrastructure for Programming Socio-Technical Systems, *ACM Transactions on Internet Technology, Special Issue on Foundations of Social Computing*, **14**(4), December 2014, 23:1–23:23, ISSN 1533-5399.

[5] Baldoni, M., Baroglio, C., Capuzzimati, F.: Typing Multi-Agent Systems via Commitments, *Post-Proc. of the 2nd International Workshop on Engineering Multi-Agent Systems, EMAS 2014, Revised Selected and Invited Papers* (F. Dalpiaz, J. Dix, M. B. van Riemsdijk, Eds.), number 8758 in LNAI, Springer, 2014, ISBN 978-3-319-14483-2, ISSN 0302-9743.

[6] Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Empowering Agent Coordination with Social Engagement, *AI*IA 2015: Advances in Artificial Intelligence, XIV International Conference of the Italian Association for Artificial Intelligence* (M. Gavanelli, E. Lamma, F. Riguzzi, Eds.), number 9336 in LNAI, Springer, Ferrara, Italy, September 2015, ISSN 0302-9743.

[7] Baldoni, M., Baroglio, C., Chopra, A. K., Desai, N., Patti, V., Singh, M. P.: Choice, Interoperability, and Conformance in Interaction Protocols and Service Choreographies, *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2009* (K. Decker, J. Sichman, C. Sierra, C. Castelfranchi, Eds.), IFAAMAS, Budapest, Hungary, May 2009, ISBN 978-0-9817381-7-8.

[8] Baldoni, M., Baroglio, C., Chopra, A. K., Singh, M. P.: Composing and Verifying Commitment-Based Multiagent Protocols, *Proc. of 24th International Joint Conference on Artificial Intelligence, IJCAI 2015* (M. Wooldridge, Q. Yang, Eds.), Buenos Aires, Argentina, July 25th-31th 2015.

[9] Baldoni, M., Baroglio, C., Marengo, E., Patti, V.: Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach, *ACM Trans. on Intelligent Sys. and Tech., Special Issue on Agent Communication*, **4**(2), March 2013, 22:1–22:25, ISSN 2157-6904.

[10] Baldoni, M., Baroglio, C., Marengo, E., Patti, V., Ricci, A.: Back to the future: An interaction-oriented framework for social computing, *First International Workshop on Requirements Engineering for Social Computing, RESC 2011, Trento, Italy, August 29, 2011*, IEEE, 2011.

[11] Bellifemine, F. L., Caire, G., Greenwood, D.: *Developing Multi-Agent Systems with JADE*, John Wiley & Sons, 2007, ISBN 0470057475.

[12] Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo, *Science of Computer Programming*, **78**(6), 2013, 747 – 761, ISSN 0167-6423, Special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) & Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.

[13] Bordini, R. H., Hübner, J. F., Tralamazza, D. M.: Using *Jason* to Implement a Team of Gold Miners, *Computational Logic in Multi-Agent Systems, 7th International Workshop, CLIMA VII, Hakodate, Japan, May 8-9, 2006, Revised Selected and Invited Papers* (K. Inoue, K. Satoh, F. Toni, Eds.), number 4371 in Lecture Notes in Computer Science, Springer, 2006.

[14] Bordini, R. H., Hübner, J. F., Wooldridge, M.: *Programming Multi-Agent Systems in AgentSpeak Using Jason*, John Wiley & Sons, 2007, ISBN 0470029005.

[15] Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., Treur, J.: Desire: Modelling Multi-Agent Systems in a Compositional Formal Framework, *Int. J. of Cooperative Information Systems*, **06**(01), March 1997, 67–94, ISSN 0218-8430.

[16] Cabac, L., Moldt, D., Rölke, H.: A Proposal for Structuring Petri Net-Based Agent Interaction Protocols, *Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands, June 23-27, 2003, Proceedings* (W. M. P. van der Aalst, E. Best, Eds.), number 2679 in Lecture Notes in Computer Science, Springer, 2003.

[17] Castelfranchi, C.: Commitments: From Individual Intentions to Groups and Organizations, *Proceedings of the First International Conference on Multiagent Systems (ICMAS)* (V. R. Lesser, L. Gasser, Eds.), The MIT Press, San Francisco, California, USA, June 1995.

[18] Cherns, A.: Principles of Socio-Technical Design, *Human Relations*, **2**, 1976, 783–792.

[19] Chesani, F., Mello, P., Montali, M., Torroni, P.: Representing and monitoring social commitments using the event calculus, *Autonomous Agents and Multi-Agent Systems*, **27**(1), 2013, 85–130.

[20] Chopra, A. K.: *Commitment Alignment: Semantics, Patterns, and Decision Procedures for Distributed Computing*, Ph.D. Thesis, North Carolina State University, 2009.

[21] Chopra, A. K., Singh, M. P.: Cupid: Commitments in Relational Algebra, *Proc. of the 29th AAAI Conf*, AAAI Press, 2015.

[22] Conte, R., Castelfranchi, C., Dignum, F.: Autonomous Norm Acceptance, *ATAL*, number 1555 in LNCS, Springer, 1998, ISBN 3-540-65713-4.

[23] Costantini, S.: Knowledge Acquisition via Non-monotonic Reasoning in Distributed Heterogeneous Environments, *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings* (F. Calimeri, G. Ianni, M. Truszczynski, Eds.), number 9345 in Lecture Notes in Computer Science, Springer, 2015.

[24] Criado, N., Argente, E., Noriega, P., Botti, V.: Reasoning about constitutive norms in BDI agents, *Logic Journal of IGPL*, **22**(1), 2014, 66–93.

[25] Damiani, F., Giannini, P., Ricci, A., Viroli, M.: Standard Type Soundness for Agents and Artifacts, *Scientific Annals of Computer Science*, **22**(2), 2012, 267–326.

[26] Dastani, M., Grossi, D., Meyer, J. C., Tinnemeier, N. A. M.: Normative Multi-agent Programs and Their Logics, *Knowledge Representation for Agents and Multi-Agent Systems, First International Workshop, KRA-MAS 2008, Sydney, Australia, September 17, 2008, Revised Selected Papers* (J. C. Meyer, J. M. Broersen, Eds.), number 5605 in Lecture Notes in Computer Science, Springer, 2008.

[27] de Brito, M., Hübner, J. F., Boissier, O.: Coupling Regulative and Constitutive Dimensions in Situated Artificial Institutions, *Multi-Agent Systems and Agreement Technologies - 13th European Conference, EUMAS 2015, and Third International Conference, AT 2015, Athens, Greece, December 17-18, 2015, Revised Selected Papers* (M. Rovatsos, G. A. Vouros, V. Julián, Eds.), number 9571 in Lecture Notes in Computer Science, Springer, 2015.

[28] Demazeau, Y.: From interactions to collective behaviour in agent-based systems, *In: Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*, 1995.

[29] Desai, N., Chopra, A. K., Singh, M. P.: Amoeba: A methodology for modeling and evolving cross-organizational business processes, *ACM Trans. Softw. Eng. Methodol.*, **19**(2), 2009.

[30] d'Inverno, M., Luck, M., Noriega, P., Rodríguez-Aguilar, J. A., Sierra, C.: Communicating open systems, *Artif. Intell.*, **186**, 2012, 38–94.

[31] Doi, T., Tahara, Y., Honiden, S.: IOM/T: an interaction description language for multi-agent systems, *4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands* (F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, M. Wooldridge, Eds.), ACM, 2005.

[32] Dunn-Davies, H. R., Cunningham, R. J., Paurobally, S.: Propositional Statecharts for Agent Interaction Protocols, *Electron. Notes Theor. Comput. Sci.*, **134**, June 2005, 55–75, ISSN 1571-0661.

[33] El-Menshawy, M., Bentahar, J., Dssouli, R.: Verifiable Semantic Model for Agent Interactions Using Social Commitments, *Languages, Methodologies, and Development Tools for Multi-Agent Systems, Second International Workshop, LADS 2009, Torino, Italy, September 7-9, 2009, Revised Selected Papers* (M. Dastani, A. E. Fallah-Seghrouchni, J. Leite, P. Torroni, Eds.), number 6039 in Lecture Notes in Computer Science, Springer, 2009.

[34] Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., Arcos, J. L.: AMELI: An Agent-Based Middleware for Electronic Institutions, *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, IEEE Computer Society, 2004.

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

[36] Fornara, N., Viganò, F., Verdicchio, M., Colombetti, M.: Artificial institutions: a model of institutional reality for open multiagent systems, *Artif. Intell. Law*, **16**(1), 2008, 89–105.

[37] Franco, M. R., Sichman, J. S.: Improving the LTI-USP Team: A New JaCaMo Based MAS for the MAPC 2013, *Engineering Multi-Agent Systems - First International Workshop, EMAS 2013, St. Paul, MN, USA, May 6-7, 2013, Revised Selected Papers* (M. Cossentino, A. E. Fallah-Seghrouchni, M. Winikoff, Eds.), number 8245 in Lecture Notes in Computer Science, Springer, 2013.

[38] Hammer, F., Derakhshan, A., Demazeau, Y., Lund, H. H.: A Multi-Agent Approach to Social Human Behaviour in Children's Play, *Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, IEEE Computer Society, 2006.

[39] Hübner, J. F., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents: "Giving the organisational power back to the agents", *Autonomous Agents and Multi-Agent Systems*, **20**, 2009.

[40] Hübner, J. F., Bordini, R. H.: Developing a Team of Gold Miners Using Jason, *Programming Multi-Agent Systems, 5th International Workshop, ProMAS 2007, Honolulu, HI, USA, May 15, 2007, Revised and Invited Papers* (M. Dastani, A. El Fallah-Seghrouchni, A. Ricci, M. Winikoff, Eds.), number 4908 in Lecture Notes in Computer Science, Springer, 2007.

[41] Hübner, J. F., Sichman, J. S., Boissier, O.: S-MOISE$^+$: A Middleware for Developing Organised Multi-agent Systems, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems, AAMAS 2005 International Workshops on Agents, Norms and Institutions for Regulated Multi-Agent Systems, ANIREM 2005, and Organizations in Multi-Agent Systems, OOOP 2005, Utrecht, The Netherlands, July 25-26, 2005, Revised Selected Papers* (O. Boissier, J. A. Padget, V. Dignum, G. Lindemann, E. T. Matson, S. Ossowski, J. S. Sichman, J. Vázquez-Salceda, Eds.), number 3913 in Lecture Notes in Computer Science, Springer, 2005.

[42] Marengo, E., Baldoni, M., Baroglio, C., Chopra, A. K., Patti, V., Singh, M. P.: Commitments with Regulations: Reasoning about Safety and Control in REGULA, *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011*, 2, IFAAMAS, Taipei, Taiwan, May 2011, ISBN 0-9826571-6-1, 978-0-9826571-6-4.

[43] Martins, R., Meneguzzi, F.: A smart home model using JaCaMo framework, *12th IEEE International Conference on Industrial Informatics, INDIN 2014, Porto Alegre, RS, Brazil, July 27-30, 2014*, IEEE, 2014.

[44] Miller, T., McGinnis, J.: Amongst First-Class Protocols, *Engineering Societies in the Agents World VIII, 8th International Workshop, ESAW 2007, Athens, Greece, October 22-24, 2007, Revised Selected Papers* (A. Artikis, G. M. P. O'Hare, K. Stathis, G. A. Vouros, Eds.), number 4995 in Lecture Notes in Computer Science, Springer, 2007.

[45] Montali, M., Calvanese, D., De Giacomo, G.: Verification of data-aware commitment-based multiagent system, *Proc. of AAMAS*, IFAAMAS/ACM, 2014.

[46] Norman, T. J., Carbogim, D. V., Krabbe, E. C. W., Walton, C. D.: Argument and multi-agent systems, in: *Argumentation Machines: New Frontiers in Argument and Computation, volume 9 of Argumentation Library*, 2003, 15–54.

[47] Odell, J., Parunak, H. V. D., Bauer, B.: Representing Agent Interaction Protocols in UML, *Agent-Oriented Software Engineering, First International Workshop, AOSE 2000, Limerick, Ireland, June 10, 2000, Revised Papers* (P. Ciancarini, M. Wooldridge, Eds.), number 1957 in Lecture Notes in Computer Science, Springer, 2000.

[48] Okouya, D., Fornara, N., Colombetti, M.: An Infrastructure for the Design and Development of Open Interaction Systems, *Post-Proc. of the 2nd International Workshop on Engineering Multi-Agent Systems, EMAS 2014, Revised Selected and Invited Papers* (M. Cossentino, A. El Fallah Seghrouchni, M. Winikoff, Eds.), number 8245 in LNAI, Springer, 2013.

[49] Omicini, A., Zambonelli, F.: TuCSoN: a Coordination model for Mobile Information Agents, *Proc. of IIIS*, IDI – NTNU, Trondheim (Norway), 8–9 June 1998, ISSN 0802-6394.

[50] Persson, C., Picard, G., Ramparany, F., Boissier, O.: A JaCaMo-Based Governance of Machine-to-Machine Systems, *Advances on Practical Applications of Agents and Multi-Agent Systems - 10th International Conference on Practical Applications of Agents and Multi-Agent Systems, PAAMS 2012, Salamanca, Spain, 28-30 March, 2012* (Y. Demazeau, J. P. Müller, J. M. C. Rodríguez, J. B. Pérez, Eds.), number 155 in Advances in Intelligent and Soft Computing, Springer, 2012.

[51] Rafaeli, S.: *Sage Annual Review of Communication Research: Advancing Communication Science: Merging Mass and Interpersonal Processes*, chapter (Chapter 4) Interactivity: From new media to communication, Sage, Beverly Hills, 1988, 110–134.

[52] Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective, *Autonomous Agents and Multi-Agent Systems*, **23**(2), 2011, 158–192.

[53] Ricci, A., Santi, A.: CArtAgO by Example, 2010, http://www.emse.fr/~boissier/enseignement/maop13/courses/cartagoByExamples.pdf.

[54] Ricci, A., Santi, A.: Typing Multi-agent Programs in simpAL, *ProMAS* (M. Dastani, J. F. Hübner, B. Logan, Eds.), number 7837 in Lecture Notes in Computer Science, Springer, 2012, ISBN 978-3-642-38699-2.

[55] Singh, M. P.: Commitments Among Autonomous Agents in Information-Rich Environments, *Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent Rationality*, Springer-Verlag, London, UK, 1997, ISBN 3-540-63077-5.

[56] Singh, M. P.: An Ontology for Commitments in Multiagent Systems, *Artif. Intell. Law*, **7**(1), 1999, 97–113.

[57] Singh, M. P.: A Social Semantics for Agent Communication Languages, *Issues in Agent Communication*, 1916, Springer, 2000.

[58] Singh, M. P.: Distributed enactment of multiagent workflows: temporal logic for web service composition, *Proc. AAMAS*, ACM, 2003.

[59] Singh, M. P.: Commitments in multiagent systems some controversies, some prospects, in: *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi* (F. Paglieri, L. Tummolini, R. Falcone, M. Miceli, Eds.), chapter 31, College Publications, London, 2011, 601–626.

[60] Singh, M. P.: Information-driven interaction-oriented programming: BSPL, the blindingly simple protocol language, *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3* (L. Sonenberg, P. Stone, K. Tumer, P. Yolum, Eds.), IFAAMAS, 2011.

[61] Telang, P. R., Singh, M. P., Yorke-Smith, N.: Relating Goal and Commitment Semantics, *ProMAS*, number 7217 in Lecture Notes in Computer Science, Springer, 2011, ISBN 978-3-642-31914-3.

[62] Thiele, A., Konnerth, T., Kaiser, S., Keiser, J., Hirsch, B.: Applying JIAC V to Real World Problems: The MAMS Case, *MATES*, number 5774 in LNCS, Springer, 2009, ISBN 978-3-642-04142-6.

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

[64] Urovi, V., Bromuri, S., Stathis, K., Artikis, A.: Initial Steps Towards Run-Time Support for Norm-Governed Systems, *Coordination, Organizations, Institutions, and Norms in Agent Systems VI - COIN 2010 International Workshops, COIN@AAMAS 2010, Toronto, Canada, May 2010, COIN@MALLOW 2010, Lyon, France, August 2010, Revised Selected Papers* (M. D. Vos, N. Fornara, J. V. Pitt, G. A. Vouros, Eds.), number 6541 in Lecture Notes in Computer Science, Springer, 2010.

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

[66] Winikoff, M., Liu, W., Harland, J.: Enhancing Commitment Machines, *Declarative Agent Languages and Technologies II, Second International Workshop, DALT 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers* (J. A. Leite, A. Omicini, P. Torroni, P. Yolum, Eds.), number 3476 in Lecture Notes in Computer Science, Springer, 2004.

[67] Yolum, P., Singh, M. P.: Designing and Executing Protocols Using the Event Calculus, *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, ACM, New York, NY, USA, 2001, ISBN 1-58113-326-X.

[68] Yolum, P., Singh, M. P.: Commitment Machines, *Intelligent Agents VIII, 8th Int. WS, ATAL 2001*, number 2333 in LNCS, Springer, 2002.

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

[70] Zatelli, M. R., Hübner, J. F.: The Interaction as an Integration Component for the JaCaMo Platform, *Post-Proc. of the 2nd International Workshop on Engineering Multi-Agent Systems, EMAS 2014, Revised Selected and Invited Papers* (F. Dalpiaz, J. Dix, M. B. van Riemsdijk, Eds.), number 8758 in LNAI, Springer, 2014.