Leveraging Commitments and Goals in Agent Interaction

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. Modeling and regulating interactions among agents is a critical step in the development of Multiagent Systems (MASs). Some recent works assume a normative view, and suggest to model interaction protocols in terms of obligations. In this paper we propose to model interaction protocols in terms of goals and commitments, and show how such a formalization promotes a deliberative process inside the agents. In particular, we take a software engineering perspective, and balance the use of commitments against obligations inside interaction protocols. The proposal is implemented via JaCaMo+, an extension to JaCaMo, in which Jason agents can interact while preserving their deliberative capabilities by exploiting commitment-based protocols, reified by special CArtAgO artifacts. The paper shows how practical rules relating goals and commitments can be almost directly encoded as Jason plans to be used as building blocks in agent programming.

Keywords: Social Computing, Agent Programming, Commitments and Goals, Agents & Artifacts, 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* [32,18,19]. 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. Currently, there are many frameworks that support designers and programmers in realizing one of these components (e.g., [8,10,25,11,29]). To the best of our knowledge, JaCaMo [9] is the the most complete among the well-established proposals, providing a thorough integration of the three components agents, environments, and organizations into a single programming framework. Another work along this direction is [7], which posits that agents and environments should be linked and interconnected through standard interfaces to fully leverage each of them.

A recent extension to JaCaMo [32] further enriches the framework by introducing an interaction component. The interaction component allows regulating

both agent interactions and the interactions between agents and environment. More precisely, an interaction component encodes –in an automaton-like shape– a *protocol*, in which states represent protocol steps, and transitions between states are associated with (undirected) *obligations* that can assume three forms: 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). Such protocols provide a *guideline* of how a given organizational goal should be achieved.

Interaction components, as defined in [32], however, present also some drawbacks. Works such as [15] show the importance, for the agents to be autonomous, to reason about the social consequences of their actions by exploiting *constitutive norms* that link the agents' actions to their respective social meanings. However, an interaction component operates as a coordinator that, by relying on obligations, issues commands about what an agent has to do, and when. This impedes agents from reasoning on the normative effects of their actions. On the one hand, the obligations are not constitutive norms while, on the other hand, the social meaning of such commands is not known to the agents but only implicitly encoded within the protocol. Agents lose part of their deliberative power since, once they join an interaction component, they have no other choice but deciding whether satisfying or not those obligations they are in charge of, while the rationale behind these obligations remains hidden to them. Consequently, this approach does not suit those situations where interaction is not subject to an organizational guideline, such as in the case when interaction is among agents and each agent decides what is best for itself [31], or when guidelines amount to declarative, underspecified constraints that still leave agents the freedom to take strategic decisions on their behavior.

Although we substantially agree with [32] about the importance of explicitly capturing the agents' interactions with appropriate abstractions, we also note that organization-driven guidelines, presented in that work, are but a kind of interaction; we thus propose a complementary approach which better supports the deliberative capabilities of the agents. Indeed, when organizational goals are not associated with corresponding guidelines, agent deliberation is crucial for the achievement of goals. An agent, in fact, has to act not only upon its own goals, but also upon what interactions could be necessary for achieving these goals. In other terms, an agent has to discover how to obtain a goal by interacting with others, i.e. to establish when to create an engagement, and which (sub)goals should be achieved first in order to fulfill its engagements. It is important to underline that when agents can fully exploit their deliberative capabilities, they can take advantage of opportunities (*flexibility*), and can find alternative ways to get their goals despite unexpected situations that may arise (*robustness*).

We claim that whenever guidelines are missing, the interactions among the agents should be supported by the very fundamental notions of goal and engagement. For this reason, we propose in this paper to complement the interaction protocol in [32], and more in general organizational and normative approaches [17,20,23,16], with an *interaction artifact* that can be used by the agents as a

common ground. Our interaction artifacts encode the notion of engagement as *social commitment* [26]. The choice of commitments stems by the fact that, differently from obligations, commitments are taken by an agent as a result of an internal deliberative process. They can be directly manipulated by the agents, and they have proved to be very effective in modeling (directed) social relationships. In addition, a recent work by Telang et al. [28] shows how goals and commitments are strongly interrelated. Commitments are therefore evidence of the capacity of an agent to take responsibilities autonomously. Citing Singh [27], an agent would become a debtor of a commitment based on the agent's own communications: either by directly saying something or having another agent communicate something in conjunction with a prior communication of the debtor. That is, there is a causal path from the establishment of a commitment to prior communications by the debtor of that commitment. By contrast, obligations can result from a deliberative process which is outside the agent; this is the case of the interaction component in [32]. This is the reason why we believe that the introduction of a deliberative process on constitutive rules that rely on obligations would not really support the agents' autonomy.

Practically, the proposal relies on the JaCaMo platform [9], and hence we dubbed it JaCaMo+: Jason agents engage commitment-based interactions which are reified as CArtAgO artifacts. CArtAgO is a framework based on the A&A meta-model [30,24] which extends the agent programming paradigm with the first-class entity of artifact: a resource that an agent can use, and that models working environments. It provides a way to define and organize workspaces, that are logical groups of artifacts, that can be joined by agents at runtime. The environment is itself programmable and encapsulates services and functionalities, making it active. JaCaMo+ artifacts represent the *interaction social state* and provide the roles agents enact. The use of artifacts enables the implementation of monitoring functionalities for verifying that the on-going interactions respect the commitments and for detecting violations and violators.

The paper extends and details the approach introduced in [3], and is organized as follows. Section 2 introduces some basic notions about goals and commitments. Section 3 discusses the extensions to JaCaMo that have been introduced in JaCaMo+; Section 4 shows, by exemplifying the FIPA Contract Net Protocol, how agents can be programmed by means of patterns encoding the interplay between goals and commitments.

2 Basic Notions

A social commitment models the directed relation between two agents: a *debtor* and a *creditor*, that are both aware of the existence of such a relation and of its current state: A commitment $\mathsf{C}(x, y, s, u)$ captures that agent x (debtor) commits to agent y (creditor) to bring about the consequent condition u when the antecedent condition s holds. Antecedent and consequent conditions are conjunctions or disjunctions of events and commitments. Unlike obligations, commitments are manipulated by agents through the standard operations *create*,

cancel, *release*, *discharge*, *assign*, *delegate* [26]. A commitment is autonomously taken by a debtor towards a creditor on its own initiative, instead of dropping from an organization, like obligations. This preserves the autonomy of the agents and is fundamental to harmonize deliberation with goal achievement. The agent does not just react to some obligations, but it rather includes a deliberative capacity by which it creates engagements towards other agents while it is trying to achieve its goals (or to the aim of achieving its goals). Since debtors are expected to satisfy their engagements, commitments satisfy the requirement in [14] of having a normative value, providing social expectations on the agents' behaviors, as well as obligations. Commitments also satisfy the requirement in [17] that when parties are autonomous, social relationships cannot but concern the *observable* behavior of the agents themselves.

Commitment-based protocols 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 commitments lifecycle. By relying on the social state, an agent can deliberate to create further commitments, or to bring about a condition involved in some existing commitment. Most importantly, commitments can be used by agents in their practical reasoning together with beliefs, intentions, and *goals*. In particular, Telang et al. [28] point out that goals and commitments are one another complementary: A commitment specifies how an agent relates to another one, and hence describes what an agent is willing to bring about for another agent. On the other hand, a goal denotes an agent's proattitude towards some condition; that is, a state of the world that the agent should achieve. An agent can create a commitment towards another agent to achieve one of its goals; but at the same time, an agent determines the goals to be pursued relying on the commitments it has towards others: A commitment is satisfied when the related goal is achieved. Note that, similarly to commitments, goals have their own lifecycle that evolves according to the actions performed by the agents (leading to the achievement or unfulfillment of goals), but also to the decisions of the agents to purse, suspend, or drop the goals themselves.

In [28], a goal G is formalized as $\mathsf{G}(x, p, r, q, s, f)$, where x is the agent pursuing G, p is a precondition that must be satisfied before G can become Active, r is an invariant condition that is true when G becomes Active and holds until the achievement of G, q is a post-condition (effect) that becomes true when G is successfully achieved, and finally, s and f are the success and failure conditions, respectively. In the following sections we will show how such a formalization can be mapped into Jason plans, and how it turns out to be useful for the agent programming purpose.

3 JaCaMo+

JaCaMo [9] is a platform integrating Jason (as an agent programming language), CArtAgO (as a realization of the A&A meta-model [30]), and Moise (as a support to the realization of organizations). In this section we shortly describe how

JaCaMo+ is obtained by extending the CArtAgO and Jason components of the standard JaCaMo.

3.1 Extending CArtAgO

Exploiting [1], JaCaMo+ enriches CArtAgO's artifacts with an explicit representation of commitments and of commitment-based protocols. The resulting class of artifacts reifies the execution of commitment-based protocols, including the social state of the interaction, and enables Jason agents both to be notified about the social events and to perform practical reasoning also about the other agents. This is possible thanks to the *social expectations* raised by commitments. Since an artifact is a programmable, active entity, it can act as a monitor of the in progress interaction. The artifact can therefore detect violations that it can ascribe to the violator without the need of agent introspection.

Specifically, a JaCaMo+ artifact encodes a commitment protocol, that is structured into a set of roles. 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 perform 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 by adding new commitments, or by modifying the state of existing commitments.

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 also add the annotation *@ROLE* to specify which roles are enabled to use that particular action. Another extension is an *explicit representation of the social state*, which is maintained within the artifact. 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 state. This is done for the sake of a greater degree of decoupling between actions/events/messages and their effects [5,6]. 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.

3.2 Extending Jason

Jason [10] 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),

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

which can be either the addition or the deletion of some belief or goal. The syntax is inherently 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. We extend the *Jason* component of JaCaMo by allowing the specification of plans whose triggering events involve commitments. JaCaMo+ represents a commitment as a term $cc(debtor, creditor, antecedent, consequent, status)$ where *debtor* and *creditor* identify the involved agents (or agent roles), while *antecedent* and *consequent* are the commitment conditions. *Status* is the commitment state (the set being defined in the commitments life cycle [21]). Commitments operations (e.g. create, see Section 2) are realized as *internal operations* of the new class of artifacts we added to CArtAgO. Thus, commitment operations cannot be invoked directly by the agents, but the protocol actions will use them as primitives to modify the social state.

A Jason plan is specified as:

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

where the *triggering_event* denotes the events 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. 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 by the addition of a commitment in the social state:

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

More precisely, the plan is triggered when a commitment, that unifies with the one in the plan head, appears in the social state. The syntax is the standard for Jason plans. *Debtor* and *creditor* are to be substituted by the proper roles. The plan may be devised so as to change the commitment status (e.g. the debtor will try to satisfy the comment), or it may be devised so as to allow the agent to react to the commitment presence (e.g., collecting information). Similar schemas can be used for commitment deletion and for the addition/deletion of social facts. Further, commitments can also be used in contexts and in plans as test goals ($?cc(\dots)$), or achievement goals ($!cc(\dots)$). Addition or deletion of such goals can, as well, be managed by plans; for example:

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

The plan is triggered when the agent creates an achievement goal concerning a commitment. Consequently, the agent will act upon the artifact so as to create the desired social relationship. After the 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.

4 Programming in JaCaMo+

In this section we show how Jason agents can be easily programmed by considering a commitment-based protocol as a guideline for the programmer: We first present a programming approach which exploits the practical rules by Telang et al. [28], and then we exemplify the approach implementing the initiator and participant agents of the well-known Contract-Net Protocol (CNP).

4.1 Practical Rules as Programming Code-Blocks

For both goals and commitments, [28] defines lifecycles, and operations through which the state of a goal, or a commitment, evolves over time. The relationship between goals and commitments is formalized in terms of *practical* rules, which capture patterns of pragmatic reasoning. They include: (1) rules from goals to commitments to capture how commitments evolve when the state of some goals change; and (2) rules from commitments to goals to capture how a goal evolves when the corresponding commitment changes in the social state. These rules can be easily encoded in JaCaMo+, and used by a programmer as templates for implementing Jason agents. In the following we discuss four examples of rules that will be used in the CNP scenario.

Goal rules. This JaCaMo+ template tackles the case when a goal $G = \mathsf{G}(x, p, r, q, s, f)$ appears in the knowledge base of agent x; namely, x wants to achieve the success condition s, and hence an appropriate plan is triggered:

```
+!G : p
  <-?r
    ⟨body⟩ /*plan achieving condition s*/
    ?q.
```

Differently from [28], in JaCaMo+ we explicitly mention a plan of actions (the body) to achieve the success condition s. When x can satisfy G autonomously (no interaction is needed), conditions s and q coincide. Instead, when x cannot satisfy G (or it is not convenient for x to achieve G autonomously), the body will involve an interaction with another agent and, as we will see, conditions q and s will differ. Note that, in JaCaMo+ we can also specify a plan to be triggered when the failure condition is reached:

```
-!G : f
  <-
    ⟨body⟩. /*plan handling failure condition f*/
```

The following three templates reflect namesake rules in [28].

Entice. Agent x can achieve G with the help of agent y: x creates an offer to agent y such that, if y brings about s (success condition of G), then x will engage into achieving a condition u of interest for y. Such an offer is naturally modeled as the commitment $C(x, y, s, u)$. The JaCaMo+ template is:

```
+!G : p
  <-?r
    social_action;
    ?cc(x, y, s, u, CONDITIONAL).
```

The body of the rule consists of a *social action*; namely, a protocol action offered by the artifact x is focused on, and whose meaning is the creation of a commitment $C = cc(x, y, s, u, CONDITIONAL)$. This commitment will push agent y to bring about the success condition s associated with G, thus this is a special case of goal activation. Note that the post condition of this rule corresponds to a test on the existence of the commitment C; agent x can verify, by inspecting the social state, that the commitment really exists.

Deliver. If commitment $C(x, y, s, u)$ becomes detached, then debtor x activates a goal $G_1 = \mathsf{G}(x, p, r, q, u, f)$ to bring about the consequent. In JaCaMo+:

```
1 +cc(x, y, s, u, DETACHED) : context
2   <- !G1 ;
3     ?cc(x, y, s, u, SATISFIED).
```

It is worth noting the test goal at the end of the rule: It allows x to verify that after the achievement of G_1, its corresponding commitment is now satisfied.

Detach. When a conditional commitment $C_1(y, x, s', t)$, appears in the social state, the creditor x activates a goal $G_2 = \mathsf{G}(x, p', r', q', s', f')$ to bring about the commitment antecedent. The JaCaMo+ template is:

```
1 +cc(y, x, s', t, CONDITIONAL) : context
2   <- !G2 ;
3     ?cc(y, x, s', t, DETACHED).
```

Note that, as in the previous case, agent x can verify that, after the satisfaction of goal G_2, the corresponding commitment is now detached.

4.2 JaCaMo+ Contract Net Protocol

As in [32], we assume that agents are assigned with institutional goals, defined in the Moise layer, to be achieved via the well-known Contract Net Protocol (CNP) protocol. We show how CNP can be implemented in JaCaMo+ by exploiting the templates introduced above. CNP (see Table 1) involves two roles: initiator (i) and participant (p). An agent playing the initiator role calls for proposals from agents playing the participant role. A participant makes a proposal if interested. Proposals can be accepted or rejected by initiator. *Accept*, *done*, and *failure* do not amount to commitment operations, but impact on the progression of commitment states, e.g., *accept* causes the satisfaction of the commitment created by *cfp*. Listing 1.1 reports an excerpt of the JaCaMo+ *CNP protocol artifact* implementation.

Table 1. CNP: actions and their social meaning.

initiator (i):	participant (p):
cfp: create(C(*i, p, propose, accept* $\vee$ *reject*))	*propose:* create(C(*p, i, accept, done* $\vee$ *failure*))
reject: release(C(*p, i, accept, done* $\vee$ *failure*))	*refuse:* release(C(*i, p, propose, accept* $\vee$ *reject*))
accept: "commitment progression"	*done:* "commitment progression"
	failure: "commitment progression"

```
1  @OPERATION
2  @ROLE(name="initiator")
3  public void cfp(String task) {
4    RoleId initiator =
5       getRoleIdByPlayerName(getOpUserName());
6    this.defineObsProperty("task", task,
7                  initiator.getCanonicalName());
8    RoleId dest = new RoleId("participant");
9    createAllCommitments(new Commitment(initiator,
10     dest,"propose","accept OR reject"));
11   assertFact(new Fact("cfp", initiator, task));
12 }
13 @OPERATION
14 @ROLE(name="participant")
15 public void propose(String prop, int cost, String init) {
16   Proposal p = new Proposal(prop, cost);
17   // ...
18   defineObsProperty("proposal",
19      p.getProposalContent(),p.getCost(),
20      participant.getCanonicalName());
21   createCommitment(new Commitment(participant,
22      initiator,"accept", "done OR failure"));
23   assertFact(new Fact("propose", participant, prop));
24   actualProposals++;
25   if (actualProposals == numberMaxProposals) {
26     // ...
27     createCommitment(new Commitment(initiator,
28               groupParticipant,"true", "accept OR reject"));
29   }
```

Listing 1.1. The CNP artifact in JaCaMo+.

cfp (line 3) is a protocol action, realized as a CArtAgO operation (CArtAgO Java annotation @OPERATION, line 1). It can be executed only by an *initiator* (JaCaMo+ Java annotation @ROLE(name="initiator"), line 2). It publishes the task for the interaction session as an observable property of the artifact (line 6). All agents focusing on the artifact will have this information added to their belief bases. The social effect of *cfp* is the creation (line 9) of as many commitments as participants to the interaction, and of a social fact (line 11), that tracks the call made by the initiator. These effects will be broadcast to all focusing agents. *Accept* pertains to the initiator. It asserts a social fact, *accept*, which causes the satisfaction of one of the commitments created at line 9 towards a specific participant. *Propose* counts the received proposals and, when their number is sufficient, signals this fact to the initiator by the creation of a commitment (line 21) towards the group of participants. Below, the *JaCaMo+* initiator program:

```
1  /* Initial goals */
2  !startCNP.
3  /* Plans */
4  +!startCNP : true
5    <-  makeArtifact("cnp","cnp.Cnp",[],C);
6        focus(C);
7        enact("initiator").
8  +enacted(Id,"initiator",Role_Id)
9    <-  +enactment_id(Role_Id);
10       !solveTask("task-one").
11 +!solveTask(Task) /*ENTICE*/
12   : enactment_id(My_Role_Id)
13   <- +task(Task);
14     cfp(Task);
15     ?cc(My_Role_Id, Part_Role_Id, "propose",
```

```
16          "(accept or reject)", "CONDITIONAL").
17 +cc(My_Role_Id, "participant", "true", /*DELIVER*/
18          "(accept OR reject)", "DETACHED")
19    : enactment_id(My_Role_Id)
20      <-!acceptORreject;
21        ?cc(My_Role_Id, _, "true",
22            "(accept OR reject)", "SATISFIED").
23 +!acceptORreject
24    : not evaluated
25      <- +evaluated;
26      .findall(proposal(Content,Cost,Id),
27          proposal(Content,Cost,Id),Proposals);
28      .count(proposal(Content,Cost,Id),
29                  ProposalsNumb);
30      .min(Proposals,
31        proposal(Proposal,Cost,Winner_Role_Id));
32      +winner(Winner_Role_Id);
33      accept(Winner_Role_Id);
34      ?cc(My_Role_Id, Winner_Role_Id,
35          "true", "(accept OR reject)", "DETACHED").
36 %... action 'reject' for all other proposals ...
37 +done(Participant_role_id, Result)
38    : winner(Participant_role_id);
39    <- .print("Task resolved: ",Result).
40 +failure(Participant_role_id)
41    : winner(Participant_role_id);
42    <- .print("Task failed by ",Participant_role_id).
```

Listing 1.2. The initiator agent code in JaCaMo+.

The first ten lines are about the setting up of the environment. In this implementation, the initiator agent first creates the Cnp artifact (line 5), and then enact the initiator role (line 7). In general, however, the artifact could already be available, and an agent could just focus on it, and enact the initiator role. Note that the artifact notifies the agent the success of the enactment by asserting an *enacted* belief in the social state; note also that the agent receives a unique identifier, Role_Id, that will be used within the social state throughout the subsequent interactions (i.e., commitments will mention such an identifier).

After these preliminary steps, the initiator tries to reach the goal of having task-one performed: solveTask("task-one")[2]. This situation maps with the *ENTICE* rule from [28]; we, thus, follow the JaCaMo+ template associated with such a rule: see lines 11 - 16. The initiator, driven by its goal, performs the social action *cfp*, and thereby creates a commitment towards any participant that is focusing (or will focus) on that specific artifact. The execution of such action (which is performed by the initiator by its own initiative) modifies the social state; consequently, this modification is notified to the other focussing agents who will be in condition of taking this new social relationship into account in their own deliberative activity. The test goal concluding the rule allows the initiator to verify that at least one commitment has actually been created; namely, the entice has changed the social state.

Since the initiator has created a commitment, it must be ready to bring about the consequent of such a commitment whenever the antecedent will become true. The initiator must therefore contain a *DELIVER*-template plan; see lines 17-22

[2] To improve the readability of the code, we have simplified the notation in the Jason program by abstracting goals with simple labels.

in which the initiator, activated by the detachment of the commitment previously created with the *cfp* social action, starts a plan that will satisfy the commitment itself. The plan, acceptORreject (lines 23-33), first selects the best proposal, and then performs a social action accept towards the winner agent, and a social action reject towards any other participant that has not been selected.

The two last plans, done (line 37) and failure (line 40), are used by the initiator to monitor the actual completion of the task with either success or failure.

Let us now consider the participant side.

```
1  /* Initial goals */
2  !participate.
3  /* Plans */
4  +!participate : true
5    <- focusWhenAvailable("cnp");
6       enact("participant").
7  +enacted(Id,"participant",My_Role_Id)
8    <- +enactment_id(My_Role_Id).
9  +cc(Initiator_Role_Id, My_Role_Id, /*DETACH*/
10     "propose", "(accept OR reject)","CONDITIONAL")
11   :enactment_id(My_Role_Id)
12      & task(Task, Initiator_Role_Id)
13   <- !setup_proposal(Task, Initiator_Role_Id);
14      ?cc(Initiator_Role_Id, My_Role_Id,
15         "true", "(accept OR reject)","DETACHED").
16 +!setup_proposal(Task, Initiator_Role_Id)
17   : enactment_id(My_Role_Id)
18   <- !prepare_proposal(Task, Prop, Cost);
19      propose(Prop, Cost, Initiator_Role_Id);
20      +my_proposal(Prop, Cost, Initiator_Role_Id);
21      ?cc(My_Role_Id, Initiator_Role_Id, "accept",
22          "(done OR failure)", "CONDITIONAL").
23 +cc(My_Role_Id, Initiator_Role_Id, /*DELIVER*/
24       "true", "(done OR failure)", "DETACHED")
25   : enactment_id(My_Role_Id) &
26      accept(My_Role_Id)
27   <- ?my_proposal(Prop, Cost, Initiator_Role_Id);
28     !doneORfailure(Prop, Cost, Initiator_Role_Id).
29     ?cc(My_Role_Id, Initiator_Role_Id,
30        "true", "(done OR failure)", "SATISFIED").
31 +!doneORfailure(Prop, Cost, Initiator_Role_Id)
32  <- !compute_result(Prop, Cost, Result);
33    if (Result == "fail"){
34     failure(Initiator_Role_Id);
35    }
36    else {
37      done(Result, Initiator_Role_Id);
38    }.
39 +!compute_result(Prop, Cost, Result)
40   <- ⟨plan computing the result⟩.
```

Listing 1.3. The participant agent code in JaCaMo+.

A participant waits for calls for proposal by means of the CArtAgO basic operation *focusWhenAvailable* (line 5). A participant, thus, must be able to react whenever a new commitment of the form $cc(initiator, participant, propose, accept \vee reject)$ pops up in the social state. This behavior corresponds to the *DETACH* template, that is encoded in the JaCaMo+ plan in lines 9-15. In particular, the participant triggers a plan, setup_proposal that will satisfy the antecedent of the commitment. Such a plan, in fact, will include the social action propose (line 19). Note that the effect of action propose is twofold: (1) it asserts a fact "propose" in the social state, and hence satisfies the antecedent of

the triggering commitment; and (2) it also creates a new commitment from the participant to the initiator (see the protocol definition in Table 1), of the form $cc(p, i, accept, done \vee failure)$. This second commitment states that the participant is committed to carry out the task in case the initiator accepts its proposal. Thus, since the participant creates a commitment, it must also be ready to bring about the consequent of that commitment when the antecedent holds, and hence also the participant has a *DELIVER*-template plan in its program: see lines 23-30. In the specific case, the participant will activate a plan, doneORfailure, whose body will include the computation of a solution for the task at hand, and also the social actions done or failure depending on the, respectively, positive or negative result of the computation.

4.3 Final Remarks

One of the strongest points of JaCaMo+ is the *decoupling* between the design of the agents and the design of the interaction – that builds on the decoupling between computation and coordination done by coordination models like tuple spaces. Agent behavior is built upon agent goals and on its engagements with other agents, which are both the result of its deliberative process. For instance, in CNP the initiator becomes active when the commitments that involve it as a debtor, and which bind it to accept or reject the proposals, are detached. It is not necessary to specify nor to manage, inside the agent, such things as deadlines or counting the received proposals: the artifact is in charge of these aspects.

The decoupling allows us to change the definition of the artifact without the need of changing the agents' implementation. The Cnp class in Listing 1.1 detaches the commitments when a certain number of proposals is received. We can substitute such a class with class CnpTimer, which detaches commitments when a given deadline expires. This modification does not have any impact on the agents, whose programs remain unchanged, but for the line in which an agent focuses on (or creates) an artifact; e.g., for the initiator, the only change occurs in line 5 (see the following listing), in which the initiator creates a different type of artifact reifing the CNP protocol (the participant case is similar).

```
1 /* Initial goals */
2 !startCNP.
3 /* Plans */
4 +!startCNP : true
5   <- makeArtifact("cnp","cnp.CnpTimer",[],C);
6      focus(C);
7      enact("initiator").
```

Listing 1.4. The initiator code, using CnpTimer.

Table 2 compares JaCaMo (with interaction [32]), with JaCaMo+ along some important characteristics that a MAS should feature. Let us discuss these dimensions, with a particular attention to those where the two platforms differ from one another. JaCaMo and JaCaMo+ do not equally support *autonomy*, in the sense that an agent can autonomously selects its own duties. JaCaMo with interaction just offers an agent to follow a predetermined path (a guideline) through which the agent has to fulfill a precise pattern of obligations. JaCaMo+, instead,

	JaCaMo with Interaction	JaCaMo+
Autonomous selection of obligations	X	✓
Maintainability	✓	✓
Monitoring Support	✓	✓
Modular Definition of Protocols	X	✓
Flexibility	X	✓
Robustness	X	✓
Interaction not spread across the agents code	✓	✓

Table 2. Comparison among JaCaMo with interaction and JaCaMo+.

offers an agent a tool, the interaction artifact, through which it can communicate with other agents and act together with others. The choice, however, of how and when been involved into an interaction remains within the scope of the agents. The adoption of commitments, in fact, assures that an agent assumes the responsibility for a task only when, by its own choice, performs a specific action on the interaction artifact. This has an impact on the property of flexibility and robustness. An interaction that is structured based on obligations only hinders agents when they need to adapt to unforeseen conditions (flexibility) or when they need to react to unwanted situations (robustness). The agent, in fact, is not free to delegate obligations, schedule them differently, etc. All the agent can do is to perform the actions that, instructed by the interaction protocol, resolve its obligations.

Protocols in [32] aim at defining guidelines to the use of resources in an organization. This, however, limits the modularity of *interaction* protocols because protocols depend on operations that are defined in the organization and there is no explicit association of which actions pertain to which roles. Thus, for instance, a participant may execute a cfp and the interaction artifact would allow it to do so. JaCaMo+ interaction protocols, instead, include the definitions of the needed operations, and specify which of them will empower the various role players. For both proposals the interaction logic is captured by the artifact and is not spread across the agent codes. Both include functionalities for monitoring the on-going interaction. In [32] the normative structure is leveraged to this aim, while in JaCaMo+ this can be done inside each of the protocol artifacts.

5 Conclusions

In this paper we presented JaCaMo+, and extension to JaCaMo that enables social behaviors into its agents. We started from the interaction protocols based on obligations proposed in [32]. These protocols are suitable for modeling interactions among different elements of a MAS (i.e., not only interactions between agents, but also between agents and objects). However, obligation-based protocols reduce agent interactions to messages that an agent is obliged to send to another agent; that is, social relationships among agents are not handled directly. In other words, an obligation-based protocol can be adopted only in an

organization that gives guidelines about how interactions should be carried on, but it is not applicable in those organizations where similar guidelines are not available.

To cope with these more challenging situations, our intuition is to define an interaction in terms of goals and commitments. Commitments, in fact, are at the right level of abstraction for modeling directed relationships between agents. Moreover, since commitments have a normative power, they enable the agents to reason about the behavior of others; a commitment creates expectations in the creditor about the behavior that the debtor will assume in the near future.

Note that our view is also backed up by the practical rules discussed in [28], which highlight how goals and commitments are each other related. In particular, in this paper we have proposed to use the same rules as a sort of methodology for programming the Jason agents. An initial implementation of our proposal is provided by the JaCaMo+ platform. The tests (which involve from 5 to 100 agents) show that it scales up quite well, despite the introduction of commitments, but a more thorough testing will be performed in the near future.

The shift from obligations to commitments is beneficial in many respects. First of all, the autonomy of the agents is better supported because, although charged with goals to be achieved, they are free in deciding how to fulfill their goals. It follows that agents are *deliberative*, and this paves the way to self-* applications, including the ability to autonomously take advantage from opportunities, and the ability 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. Thus, our concluding claim is that directly addressing social relationships increases the robustness of the whole MAS.

In the future, we intend to investigate how agents can leverage on their deliberative capabilities, and use it not only to program interactions, but to plan social interactions. Moreover, the modular nature of the implementation facilitates the development of extensions for tackling richer, data-aware contexts [12,22,13]. 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 [4], as well as to introduce a typing system along the line of [2].

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments, which helped improving the paper.

References

1. Matteo Baldoni, Cristina Baroglio, and Federico Capuzzimati. Social computing in JaCaMo. In *Proc. of ECAI*, volume 263 of *Frontiers in Artificial Intelligence*

and Applications, pages 959–960. IOS Press, 2014.
2. Matteo Baldoni, Cristina Baroglio, and Federico Capuzzimati. Typing Multi-Agent Systems via Commitments. In F. Dalpiaz, J. Dix, and M. B. van Riemsdijk, editors, *Post-Proc. of the 2nd International Workshop on Engineering Multi-Agent Systems, EMAS 2014, Revised Selected and Invited Papers*, number 8758 in LNAI, pages 388–405. Springer, 2014.
3. Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. Programming with Commitments and Goals in JaCaMo+ (Extended Abstract). In *Proc. of 14th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015*, July 4th-8th 2015.
4. Matteo Baldoni, Cristina Baroglio, Amit K. Chopra, and Munindar P. Singh. Composing and Verifying Commitment-Based Multiagent Protocols. In M. Wooldridge and Q. Yang, editors, *Proc. of 24th International Joint Conference on Artificial Intelligence, IJCAI 2015*, Buenos Aires, Argentina, July 25th-31th 2015.
5. Matteo Baldoni, Cristina Baroglio, Elisa Marengo, and Viviana Patti. Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach. *ACM Trans. on Intelligent Sys. and Tech., Special Issue on Agent Communication*, 4(2):22:1–22:25, March 2013.
6. Matteo Baldoni, Cristina Baroglio, Viviana Patti, and Elisa Marengo. Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach (Extended Abstract). In M. Wooldridge and Q. Yang, editors, *Proc. of 24th International Joint Conference on Artificial Intelligence, IJCAI 2015*, Buenos Aires, Argentina, July 25th-31th 2015.
7. Tristan M. Behrens, Koen V. Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, 2011.
8. Fabio L. Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.
9. Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747 – 761, 2013.
10. Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, 2007.
11. Frances M. T. Brazier, Barbara M. Dunin-Keplicz, Nick R. Jennings, and Jan Treur. Desire: Modelling Multi-Agent Systems in a Compositional Formal Framework. *Int. J. of Cooperative Information Systems*, 06(01):67–94, March 1997.
12. Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. Representing and monitoring social commitments using the event calculus. *Autonomous Agents and Multi-Agent Systems*, 27(1):85–130, 2013.
13. Amit K. Chopra and Munindar P. Singh. Cupid: Commitments in relational algebra. In *Proc. of the 29th AAAI Conf*, pages 2052–2059. AAAI Press, 2015.
14. Rosaria Conte, Cristiano Castelfranchi, and Frank Dignum. Autonomous Norm Acceptance. In *ATAL*, volume 1555 of *LNCS*, pages 99–112. Springer, 1998.
15. Natalia Criado, Estefania Argente, Pablo Noriega, and Vicent Botti. Reasoning about constitutive norms in BDI agents. *Logic Journal of IGPL*, 22(1):66–93, 2014.
16. Natalia Criado, Estefania Argente, Pablo Noriega, and Vicent Botti. Reasoning about norms under uncertainty in dynamic environments. *International Journal of Approximate Reasoning*, 2014.
17. Mehdi Dastani, Davide Grossi, John-Jules Ch. Meyer, and Nick A. M. Tinnemeier. Normative Multi-agent Programs and Their Logics. In *KRAMAS*, volume 5605 of *LNCS*, pages 16–31. Springer, 2008.

18. Yves Demazeau. From interactions to collective behaviour in agent-based systems. In *In: Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*, 1995.
19. Frodi Hammer, Alireza Derakhshan, Yves Demazeau, and Henrik Hautop Lund. A multi-agent approach to social human behaviour in children's play. In *Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology*, pages 403–406. IEEE Computer Society, 2006.
20. Felipe Meneguzzi and Michael Luck. Norm-based behaviour modification in BDI agents. In *AAMAS (1)*, pages 177–184. IFAAMAS, 2009.
21. Felipe Meneguzzi, Pankaj R. Telang, and Munindar P. Singh. A first-order formalization of commitments and goals for planning. In *AAAI*. AAAI Press, 2013.
22. Marco Montali, Diego Calvanese, and Giuseppe De Giacomo. Verification of data-aware commitment-based multiagent system. In *Proc. of AAMAS*, pages 157–164. IFAAMAS/ACM, 2014.
23. Daniel Okouya, Nicoletta Fornara, and Marco Colombetti. An infrastructure for the design and development of open interaction systems. In M. Cossentino, A. El Fallah Seghrouchni, and M. Winikoff, editors, *Post-Proc. of the 2nd International Workshop on Engineering Multi-Agent Systems, EMAS 2014, Revised Selected and Invited Papers*, number 8245 in LNAI, pages 215–234. Springer, 2013.
24. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *JAAMAS*, 17(3):432–456, 2008.
25. Andrea Omicini and Franco Zambonelli. TuCSoN: a coordination model for mobile information agents. In *Proc. of IIIS*, pages 177–187. IDI – NTNU, Trondheim (Norway), 8–9 June 1998.
26. Munindar P. Singh. An ontology for commitments in multiagent systems. *Artif. Intell. Law*, 7(1):97–113, 1999.
27. Munindar P. Singh. Commitments in multiagent systems some controversies, some prospects. In Fabio Paglieri, Luca Tummolini, Rino Falcone, and Maria Miceli, editors, *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi*, chapter 31, pages 601–626. College Publications, London, 2011.
28. Pankaj R. Telang, Neil Yorke-Smith, and Munindar P. Singh. Relating Goal and Commitment Semantics. In *Proc. of ProMAS*, volume 7212 of *LNCS*, pages 22–37. Springer, 2012.
29. Alexander Thiele, Thomas Konnerth, Silvan Kaiser, Jan Keiser, and Benjamin Hirsch. Applying JIAC V to Real World Problems: The MAMS Case. In *MATES*, volume 5774 of *LNCS*, pages 268–277. Springer, 2009.
30. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *JAAMAS*, 14(1):5–30, 2007.
31. Pinar Yolum and Munindar P. Singh. Commitment Machines. In *Intelligent Agents VIII, 8th Int. WS, ATAL 2001*, volume 2333 of *LNCS*, pages 235–247. Springer, 2002.
32. Maicon R. Zatelli and Jomi F. Hübner. The Interaction as an Integration Component for the JaCaMo Platform. In F. Dalpiaz, J. Dix, and M. B. van Riemsdijk, editors, *Post-Proc. of the 2nd International Workshop on Engineering Multi-Agent Systems, EMAS 2014, Revised Selected and Invited Papers*, number 8758 in LNAI, pages 431–450. Springer, 2014.