Reasoning about Social Relationships with Jason

Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati

Università degli Studi di Torino — Dipartimento di Informatica
c.so Svizzera 185, I-10149 Torino (Italy)
{matteo.baldoni,cristina.baroglio,federico.capuzzimati}@unito.it

Abstract. This work faces the problem of enabling an approach to agent programming, which allows agents to seamlessly manage and work both on social relationships and on abstractions which typically characterize agents themselves, like goals, beliefs, intentions. A similar approach is necessary in order to easily develop Socio-Technical Systems and provides a basis for carrying on methodological studies on system engineering. The paper presents an extension of JaCa(Mo) in which Jason agents can reason on social relationships, that are represented as commitments, and where Jason agents interact by way of special CArtAgO artifacts, which reify commitment-based protocols.

Keywords: Commitments, Commitments, Agents and Artifacts, Social Relationships, Social Middleware, Jason, Implementation

1 Introduction

Socio-technical systems (STS) [34,14,32] support human users by mechanizing processes and by aiding stakeholders in interacting with each other, e.g. for contending resources, asking for co-working activities, or for assigning sub-units of work. STS can be viewed as an evolution of computing [36] into *social computing* [18], with a transition from an individualistic to a societal view, where notions like *social structure*, *role* and *norm* come into play. They are organized in a set of stacked layers (see Figure 1), the lowest concerning physical resources (the equipment), the others concerning higher and higher abstractions and functionalities (e.g. communication, application, business process). The topmost concerns the laws and the regulations that govern the operation of the system (society).

Traditional approaches to software engineering do not fit the needs of STS, which amount to large-scale, multi-party, cross-organizational systems, especially those needs which concern the design of *interaction* and *distributed computation* [32]. A common way to model user activities is by means of *business processes*. Business processes, in their classic definition, are sets of structured, ordered tasks. This approach does not naturally fit STS, one of whose main characteristics is the *autonomy* of the involved parties, because business processes usually also *hard code* the regulations of the system. As a consequence, regulations maintenance requires a modification of the components behavior and of the business processes. The drawback is that system modularity is reduced and, in order to upgrade the system, it is necessary that the components code be available. Our claim is that not only there is the need for an explicit representation of the norms that rule interaction in a system but that such layer should be integrated in the system in the form of *resources* that are available to system components. That is to say, the society level of STS should be realized by way of first-class objects as well as the system components. Modeling them as resources, that are part of the system at the same level of the system components, allows to dynamically recognize, accept, refuse, modify, manipulate them, and decide whether to conform to them, as advised in [17].

Many authors, e.g. [31,27,12], argue that a better way to model STS is by way of *Multi-Agent Systems*. However, current frameworks and platforms, e.g. [7,10,26,11,33], do not account for the *social aspects* of interaction. Again, they dissolve STS topmost level into the business process level, spreading the interaction logic across the agents' implementations. As before, the drawback is that the autonomy of the parties is not preserved. Other proposals [24,31] account for the social aspects of interaction but do not consider them as system resources.

This work provides an agent framework that includes the missing social level, modeled by way of commitment-based interaction protocols [29,30] and where commitments amount to social relationships among the parties. Following the A&A meta-model [35,25] the social level is reified in a resource, that can be used, observed, manipulated by the agents that are involved in the interaction. Practically, the proposal relies on the JaCaMo platform [8]: Jason agents interact by way of commitment-based interaction protocols which are reified as CArtAgO artifacts. Such artifacts represent the 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.

2 Modeling Social Relationships

In the 50's, the Tavistock Institute of Human Relations developed a model of work organization, characterized by the assignment of responsibilities, the decentralization of work structures, the grouping of employees into mindful teams. The model is called *socio-technical design* of *socio-technical systems* [34]. Inspired by [32], we see STS as structured in three architectural macro-level (Figure 1): *infrastructural*, *functional* and *socio-organizational*. The first concerns hardware equipment, software and data management; the second organizes activities, involving actors and components; the third regulates and norms how actors can use the system, their constraints, and the laws that rule the system.

Most of the Multi-Agent frameworks and platforms mix the third level with the second, rather than explicitly accounting for a social layer of interaction, and, thus, spread the interaction logic across the agents' implementations. Consider, for instance, the well-known JADE [7] and Jason [10], which respectively rely on the object-oriented and on the declarative paradigms, and the way they

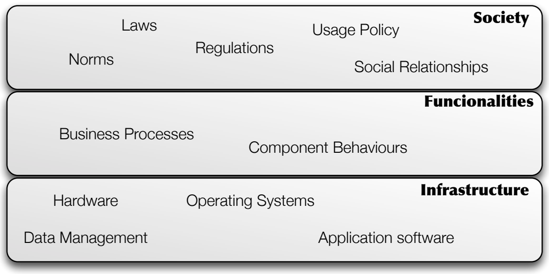

Fig. 1. Architectural macro-levels in a STS.

manage a simple interaction like the one captured by FIPA Contract Net Protocol (CNP). In the JADE example implementation [7, Sec. 5.4, page 100-107], agents execute one of two behavior, named respectively *ContractNetInitiator* and *ContractNetResponder*, that are the projections of the two roles of FIPA CNP. These include the interaction rules and contain all the checks related to the flow of messages, implemented based on finite state machines (*FSMBehavior*). These behaviors are provided by the package *jade.proto*. In Jason [10, Sec. 6.3, page 130], the interaction rules are again split into the behaviors of the interacting agents. Also in this case, and despite the declarative nature of Jason, each behavior contains a collection of plans, one for each of the interaction steps of the protocol (like *cfp* and *propose*).

We propose to explicitly represent the third level of STS by representing *social relationships* among the agents, i.e normatively defined relationships and the expected patterns of interaction between two or more agents, resulting from possession of the roles, and subject to social control by monitoring the observable behavior. We envisage both agents and social relationships as first-class entities that interact in a bi-directional manner. Social relationships are created by the execution of *interaction protocols* and provide expectations on the agents' behavior. It is, therefore, necessary to provide the agents the means to create and to manipulate, and to observe, to reason and to deliberate on social relationships so to take proper decisions about their behavior.

Specifically, we model social relationships as *commitments* [29]. A commitment $\mathsf{C}(x, y, r, p)$ captures that the agent x (debtor) commits to the agent y (creditor) to bring about the consequent condition p when the antecedent condition r holds. Antecedent and consequent conditions are conjunctions or disjunctions of events and commitments. When r equals $\top$, we use the short notation $\mathsf{C}(x, y, p)$. Commitments have a *regulative* nature, in that debtors are expected to behave so as to satisfy the engagements they have taken. This practically means that an agent is expected to behave so as to achieve the consequent conditions of the active commitments of which it is the debtor. Commitments satisfy the requirement in [19] that in a system made of autonomous and heterogeneous actors, social relationships cannot but concern the observable behavior . On the other hand, they also satisfy the requirement in [17] of having a normative value, consequently providing social expectations on the stakeholders' behavior. As a consequence, they can be used by agents in their practical reasoning together with beliefs, intentions, and goals.

Commitments are manipulated by means of the standard operations *create*, *cancel*, *release*, *discharge*, *assign*, *delegate*. As in [29], we postulate that discharge is performed concurrently with the actions that lead to the given condition being satisfied and causes the commitment to not hold. Delegate and assign transfer commitments respectively to a different debtor and to a different creditor. The interacting agents share a *social state* that contains commitments, and affect it while performing their activities. In particular, a *commitment-based interaction protocols* is a collection of actions, whose social effects are expressed in terms of standard commitment operations [37,15].

In our proposal, the social state and the commitment-based interaction protocol together define the third level of an STS, accounting for the societal regulations which rule its functioning. The social relationships (commitments) are reified as resources that are made available to the interacting agents, in the very same way as other kinds of resources of an STS [32]. We do so by relying on *artifacts*. The *Agents and Artifacts* (A&A) meta-model [35,25] extends the agent paradigm with the *artifact* primitive abstraction. An artifact is a computational, programmable system resource, that can be manipulated by agents. The A&A paradigm provides ways for defining and organizing workspaces, i.e. logical groups of artifacts, that can be joined by agents at run-time and where agents can create, use, share artifacts to support their activities.

We interpret the fact that an agent uses an artifact as the explicit acceptance, by the agent, of the norms encoded by that artifact, and modeled by the interaction protocol that the artifact reifies. The agent declares that will behave in a way that complies with the protocol. The artifact can act as a monitor of the interaction because the interaction is performed through its roles, and detect violations that it can ascribe to the violator without the need of agent introspection. Instead, in solutions that hard code the interaction rules, the check necessarily requires agent introspection. Explicit acceptance of the interaction rules is important also because it allows the interacting parties to perform *practical reasoning*, based on expectations: participants expect that the debtors of commitments behave so as to satisfy the corresponding consequent conditions.

3 Programming social relationships in Jason

JaCaMo [8] is a platform integrating Jason (as an agent programming language), CArtAgO (as a realization of the A&A meta-model), and Moise (as a support to the realization of organizations). A MAS realized in JaCaMo is a Moise agent organization, which involves a set of Jason agents, all working in shared distributed artifact-based environments, programmed in CArtAgO. CArtAgO environments can be designed and programmed as a dynamic set of artifacts, possibly distributed among various nodes of a network, that are collected into workspaces. By *focusing* on an artifact, an agent registers to be notified of events that are generated inside the artifact, e.g. when other agents execute some action.

Jason [10] implements in Java, and extends, the agent programming language AgentSpeak(L). Agents are characterized by a BDI architecture. Each of them has an own belief base, which is a set of ground (first-order) atomic formulas. Each has an own set of plans (plan library). It is possible to specify two types of goals: achievement and test goals. An achievement goal (given as an atomic formula prefixed by the '!' operator) states that the agent wants to reach a state where the specified formula holds. A test goal (prefixed by '?') states that the agent wants to test whether the associated atomic formula can be unified with one of its beliefs. Agents can reason on their beliefs/goals and react to events, amounting either to belief changes (occurred by sensing their environment) or to goal changes. Each plan has a triggering event (an event that causes its activation), 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 some agent of the MAS on the CArtAgO environment, whose consequences are automatically propagated.

However, JaCaMo does not provide any support to the realization of the STS social level. We need to introduce a way to reify the social relationships and for turning changes to the social state into events on which agents can reason and that may trigger the execution of plans. We did so by extending the CArtAgO component with libraries for realizing artifacts that implement commitment-based interaction protocols. The aim is to integrate Jason BDI with social commitments. To this purpose, we leverage the connection between Jason and CArtAgO, offered by JaCaMo, to model a class of artifacts that reify the execution of commitment-based protocols, including their social state, and that enable practical reasoning about social expectations, by means of commitments. A commitment is represented as a term *cc(debtor, creditor, antecedent, consequent, status)*, where *debtor* and *creditor* are the identities of the involved agents, while *antecedent* and *consequent* are the commitment conditions: the *debtor* is responsible towards the *creditor* agent for the satisfaction of commitment. *Status* is a further parameter that we use to keep track of the commitment state. Following [23] the states of the commitment life cycle are: created, satisfied, violated, conditional, detached, expired, pending, terminated.

The social state of the ongoing interaction is mapped onto the belief base of the agents which are focusing on the artifact: any modification of the former is instantaneously propagated to the latter. The artifact is responsible for maintaining the social state up-to-date, depending on the actions that are executed. It updates the state of commitments, according to the commitment life cycle. It provides such information to the focusing agents, by exploiting proper *observable properties*, that are added to or removed from the artifact properties.

A protocol action is implemented as an artifact operation; its execution causes the update of the social state, by adding new commitments or by modifying the state of existing commitments. An agent that is focusing on an artifact can execute protocol actions that are provided by the artifact itself: they can be executed by the agent, if the enacted role matches with the role to which the action is associated, otherwise the execution produces a failure. The check is transparent to the agent.

We extend the language by allowing plan specifications whose triggering events involve commitments, similarly to what done with beliefs. Commitments can also be used inside a plan context or body. As a difference with beliefs commitment assertion/deletion can only occur through the artifact. For example, this is the case that deals with commitment addition:

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

The plan is triggered when a commitment that unifies with the one on the left hand side appears in the social state with the specified status. The syntax is the standard for Jason plans. *Debtor* and *Creditor* are to be substituted by the proper role names. A similar schema can be used in the case of commitment deletion and in the case of addition (deletion) of social facts. 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:

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

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.

For clarity, we report a couple of examples. The following is the plan by which an agent, playing the role Participant, tries to achieve the goal of creating the conditional commitment that in case its proposal is accepted, it will complete the task unless some failure occurs.

```
+!cc(My_Role_Id, Initiator_Role_Id, "accept",
                 "(done OR failure)", "CONDITIONAL")
  :   enactment_id(Role_Id) &
      task(Task, Initiator_Role_Id)
  <-  !prepare_proposal(Task, Prop, Cost);
      propose(Prop, Cost, Initiator_Role_Id);
      +my_proposal(Prop, Cost, Initiator_Role_Id).
```

My_Role_Id unifies with the belief *enactment_id(My_Role_Id)*, added as consequence of a successful enactment. The plan context is that the agent actually enacted the role Participant, and that the agent knows the task to be solved –such knowledge is propagated by the artifact thanks to an observable property. This is, instead, the plan that is triggered when the commitment that is created, when the previous plan succeeds, changes state and becomes detached.

```
+cc(My_Role_Id,Initiator_Role_Id,"true","(done OR failure)","DETACHED")
 :   enactment_id(My_Id) & accept(My_Role_Id)
 <-  ?my_proposal(Prop, Cost, Init_Id);
     !compute_result(Prop, Cost, Result);
     if (Result == "fail") { failure(Init_Id); }
     else { done(Result, Init_Id); }.
```

The realized JaCaMo extension tis inspired by *2COMM* [2], which provides a bridge between CArtAgO and JADE for realizing commitment protocols. From an organizational perspective, a commitment protocol is structured into a set of roles, representing different ways of manipulating the social state. By enacting a role, an agent receives social powers by the artifact, whose execution has public social consequences, expressed in terms of commitments. Figure 2 reports a UML

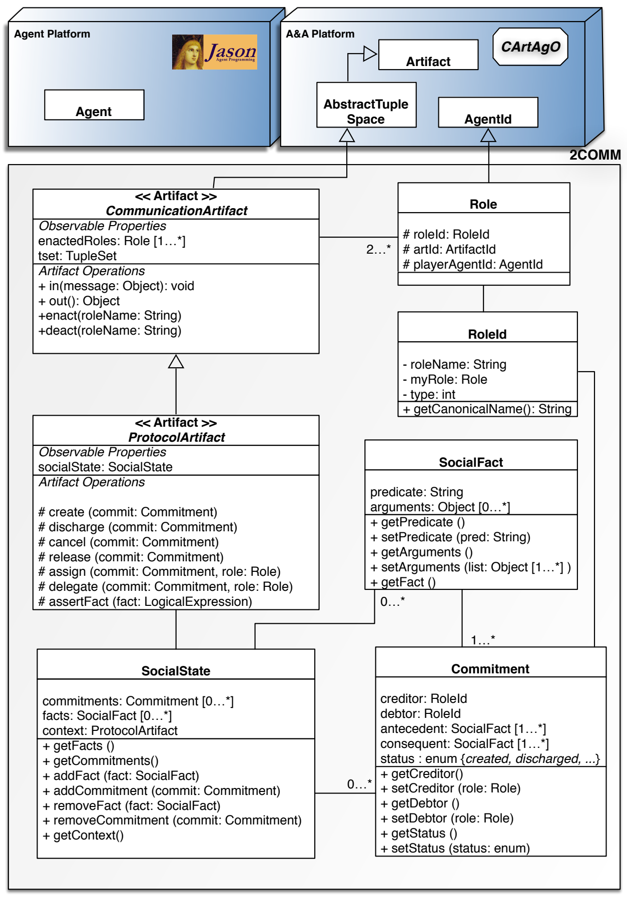

Fig. 2. UML diagram of an excerpt of the JaCaMo extension.

diagram of the libraries that were realized for extending JaCaMo. The main classes are:

– Class *CommunicationArtifact* (CA for short) provides the basic communication operations *in* and *out* for allowing mediated communication. CA extends an abstract version of the *TupleSpace* CArtAgO artifact. CA also traces who is playing which role by using the property *enactedRoles*. CA also provides the operation *enact* which allows an agent to play a role. If the enactment is successful, the artifact broadcasts the corresponding information to the agents which focused on it.

– *ProtocolArtifact* (PA for short) extends CA and allows modeling the social layer with the help of commitments. It maintains the state of the ongoing protocol interaction, via the property *socialState*, a store of social facts and commitments, that is managed only by its container artifact. This artifact implements the operations needed to manage commitments (create, discharge, cancel, release, assign, delegate). PA realizes the commitment lifecycle and the assertion/retraction of facts. Operations on commitments are realized as *internal operations*, that is, they cannot be invoked directly: the protocol social actions will use them as primitives to modify the social state.

A *commitment-based interaction protocol* is an extension of PA which defines the proper social and communicative actions as operations on the artifact itself. Actions can have guards that correspond to *context preconditions*: each such condition specifies the context in which the respective action produces the described social effect. Following the CArtAgO syntax, protocol actions are tagged with the Java annotation *@OPERATION*; this marks a public operation that agents can invoke on the artifact. Thus, for protocol artifacts a method tagged with *@OPERATION* corresponds to a social action. Our proposal adds the annotation, *@ROLE*, that specifies which role(s) is (are) allowed to use that particular action: this is the way in which we associate powers to the corresponding roles.

4 The Contract Net Protocol

In order to explain how to model social relationships with artifacts, and how to use them in Jason agents, let us sketch a commitment-based implementation of the well-known Contract Net Protocol. The reference protocol is:

cfp **means** create(*C(initiator, participant, propose, accept* $\vee$ *reject)*)
accept **means** *none*
reject **means** release(*C(participant, initiator, accept, done* $\vee$ *failure)*)
propose **means** create(*C(participant, initiator, accept, done* $\vee$ *failure)*)
refuse **means** release(*C(initiator, participant, propose, accept* $\vee$ *reject)*)
done **means** *none*
failure **means** *none*

none means that the only social consequence of the action is that it occurred. The implementation is composed by the protocol artifact, that reifies the protocol itself and that maintains the evolution of the interaction, and by the agents which enact the protocol roles, and use the artifact to interact. This is an excerpt of the protocol artifact:

```
public class Cnp extends ProtocolArtifact {
  private int numberMaxProposals = 10;
  private int actualProposals = 0;
  @OPERATION
  @ROLE(name="initiator")
  public void cfp(String task) {
    RoleId initiator =
       getRoleIdByPlayerName(getOpUserName());
    this.defineObsProperty("task", task,
              initiator.getCanonicalName());
    RoleId dest = new RoleId("participant");
    createAllCommitments(new Commitment(initiator, dest,
      "propose", "accept OR reject"));
    assertFact(new Fact("cfp", initiator, task));
  }
  @OPERATION
  @ROLE(name="initiator")
  public void accept(String participant) {
    RoleId participant =
           getRoleIdByRoleCanonicalName(participant);
    assertFact(new Fact("accept"),
              participant.getCanonicalName());
  }
  @OPERATION
  @ROLE(name="participant")
  public void propose(String prop, int cost, String init) {
    Proposal p = new Proposal(prop, cost);
    RoleId participant = getRoleIdByPlayerName(getOpUserName());
    RoleId initiator = getRoleIdByRoleCanonicalName(init);
    p.setRoleId(participant);
    defineObsProperty("proposal", p.getProposalContent(),
              p.getCost(), participant.getCanonicalName());
    createCommitment(new Commitment(participant, initiator,
              "accept", "done OR failure"));
    assertFact(new Fact("propose", participant, prop));
    actualProposals++;
    if (actualProposals == numberMaxProposals) {
      RoleId groupParticipant = new RoleId("participant");
      createCommitment(new Commitment(initiator, groupParticipant,
              "true", "accept OR reject"));
    }
  // ... other protocol operations ...
}
```

The operation *cfp* (identified by the CArtAgO Java annotation @OPERATION) is a social action which can be executed only by an agent playing the role *initiator* (Java annotation @ROLE(name="initiator") in our implementation). It is used to publish the task for the interaction session as an observable property of the artifact (*this.defineObsProperty("'task", task, initiator.getCanonicalName())*. All agents which focus on the artifact will have this information automatically added to their belief base. The social effects of *cfp* are the creation of as many commitments (*createAllCommitments(new Commitment(initiator, dest, "propose", "accept OR reject")))* as participants to the interaction, and of a social fact (*assertFact(new Fact("cfp", initiator, task))*). Similarly as above also these effects will be added to the agents' belief bases. The operation *accept* requires the initiator role. It asserts a social fact, *accept*, which causes the satisfaction and the deletion of the previous commitment towards a specific participant. The operation *propose* counts the received proposals and, when their number is sufficient, signals this fact to the initiator by the creation of a commitment (*Commitment(initiator, groupParticipant, "true", "accept OR reject")*) towards the group of participants.

We now report and comment excerpts of Jason agents code. Let us begin with the *Initiator*:

```
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       !cc(Role_Id, "participant", "propose",
11              "(accept OR reject)","CONDITIONAL").
12 +!cc(My_Role_Id, "participant", "propose",
13              "(accept OR reject)","CONDITIONAL")
14   <-  .print("sending cfp");
15       .wait(2000);
16       cfp("task-one").
17 @commit[atomic]
18 +cc(My_Role_Id, "participant", "true",
19              "(accept OR reject)", "DETACHED")
20   :  enactment_id(My_Role_Id) & not evaluated
21   <-  +evaluated;
22       .wait(2000);
23       .findall(proposal(Content,Cost,Id),
24              proposal(Content,Cost,Id),Proposals);
25       .min(Proposals,Winner);
26       +winner(Winner);
27       ?winner(proposal(Proposal,Cost,Winner_Role_Id));
28       accept(Winner_Role_Id).
29       ... action 'reject' for all other proposals ...
30 +cc(Participant_Role_Id, My_Role_Id, "true",
31              "(done OR failure)", "DISCHARGED")
32   :  done(Result)
33   <-  .print("Task resolved: ",Result).
34 +cc(Participant_Role_Id, My_Role_Id, "true",
35              "(done OR failure)", "DISCHARGED")
36   :  failure(Participant_Role_Id)
37   <-  .print("Task failed by ",Participant_role_id).
```

!startCNP, line 2, is an initial goal, that is provided for beginning the interaction. In this implementation, the agent which plays the initiator role is in charge for creating the artifact (*makeArtifact("cnp","cnp.Cnp",[],C)*) that will be used for the interaction. The agent will, then, enact the role "initiator" (*enact("initiator")*); the artifact will notify the success of the operation by asserting an *enacted* belief. Since the program contains the plan triggered by the *enacted* belief, the initiator agent can, then, execute *cfp*. When enough participants will have committed to perform the task, in case their proposal is accepted (*cc(My_Role_Id, "participant", "true", "(accept OR reject)","DETACHED")*), the initiator agent evaluates the proposals and decides which to *accept* (we omit the reject case for sake of brevity). Accepting a proposal is an action offered by the CNP artifact; it will update the social state according to the social effects devised for the action. For the *participant* agent:

```
1  /* Initial goals */
2  !participate.
3  /* Plans */
4  +!participate : true
5    <-  focusWhenAvailable("cnp");
6        enact("participant").
7  +enacted(Id,"participant",Role_Id)
```

```
  <-  +enactment_id(Role_Id).
+cc(Initiator_Role_Id, My_Role_Id, "propose",
              "(accept OR reject)","CONDITIONAL")
  :  enactment_id(My_Role_Id)
  <- !cc(My_Role_Id, Initiator_Role_Id, "accept",
              "(done OR failure)", "CONDITIONAL").

+!cc(My_Role_Id, Initiator_Role_Id, "accept",
              "(done OR failure)", "CONDITIONAL")
  :   enactment_id(Role_Id) &
      task(Task, Initiator_Role_Id)
  <-  !prepare_proposal(Task, Prop, Cost);
      propose(Prop, Cost, Initiator_Role_Id);
      +my_proposal(Prop, Cost, Initiator_Role_Id).
+cc(My_Role_Id, Initiator_Role_Id, "true",
              "(done OR failure)", "DETACHED")
  :   enactment_id(My_Id) & accept(My_Role_Id)
  <-  ?my_proposal(Prop, Cost, Init_Id);
      !compute_result(Prop, Cost, Result);
      if (Result == "fail")
      {
        failure(Init_Id);
      } else {
        done(Result, Init_Id);
      }.
+cc(My_Role_Id, Initiator_Role_Id, "true",
              "(done OR failure)", "DETACHED")
  :    enactment_id(My_Role_Id) & not accept(My_Id)
  <-  .print("proposal rejected").
+!prepare_proposal(Task,Prop,Cost)
  <-  .my_name(My_Id);
      .concat("proposal-",My_Id,Prop);
      .random(C);
      Cost = math.round(C*100).
+!compute_result(Prop, Cost, Result)
  <-  .random(Succ);
      if (Succ < 0.5) {
        .concat("fail",Result);
      } else {
        .concat(Prop,"-",Cost,"-done",Result);
      }.
```

A participant will wait for calls for proposal by means of the CArtAgO basic operation *focusWhenAvailable*. As for the initiator, the participant agent reacts to the commitment by the initiator agent, to accept or reject (a proposal), by preparing a proposal or sending a refusal – we omit details for the sake of brevity. When the participant realizes that the initiator has accepted a proposal, it checks whether its proposal was accepted or rejected, behaving accordingly.

5 Discussion and Conclusions

Following [31], social relationships provide the invariants of an STS, and govern the behavior of the peers taking part into it. The ability of the agents of reasoning on the social relationships is, thus, a fundamental element of STS. In order to enable it, it is necessary to turn social relationships into system resources, that can be handled by the participants to an interaction. In other words, social relationships are to be considered as first-class abstractions, at the same level of the other abstractions used by the agents in their reasoning. Nowadays, agent-oriented software engineers can choose from a substantial number of agent platforms, e.g. [22,9,20,5]. The choice is related to heterogeneous factors, like: scope and purpose of the system; formal model the platform is based on; richness of the agent programming language (APL), if devised; platform support, maintenance and update; (graphical) tools for supporting design and development; simplicity of integration with other (agent) programming languages. However, while platforms and frameworks like JADE [6], TuCSoN [26], DESIRE [11], JIAC [33] provide coordination mechanisms and communication infrastructures [9], they are not adequate to the purpose of realizing STS because they do not account for the social level of the interaction.

The paper presented an extension of JaCaMo with a set of libraries which realize the social level of STS, based on commitments and commitment-based protocols. Jason agents are automatically notified of the updates to the social state, and can reason about such events and trigger action plans exactly as they do for beliefs, goals and the like. The advantages of the proposal are many. First of all, thanks to the presence of reified interaction protocols, (1) agents result being loosely coupled and (2) it is not necessary to hard code the logic of interaction inside each of them. In other words, the protocol implementation is not distributed inside the agents' code but the protocol is a separate resource. Agents use their own behaviors to interact through the protocol resource, of course in a way that complies with the protocol requirements. Protocols can be modified independently from agents. As a consequence the system maintainability is increased and the autonomy of the agents is preserved.

We have already shown that this is not true in the case of JADE implementations and also for Jason [10, Sec. 6.3, page 130]. The use of JaCaMo, partly overcomes these drawbacks by way of the artifact, which provides a communication board the agents can use. Protocol actions are clearly defined as public operations, making the definition of protocols easier and modular. Still, an explicit notion of social relationship, that binds the interacting agents and thus allows a form of reasoning that is based on social expectations, is missing. For instance, [28, page 52] reports the plan:

```
+task(Task,CNPBoard) : task_descr(Task)
  <- println("found a task: ",Task);
      lookupArtifact(CNPBoard,BoardId);
      focus(BoardId);
      !make_bid(Task,BoardId).
```

the participant sends a bid after the assertion of a belief *task*$(Task, CNPBoard)$ in the artifact, automatically propagated to its belief state. In other words, the agent decides how to act merely on the basis of its own beliefs. If we consider the analogous step in our implementation (lines 9-21), the bidding is performed only as a consequence of a commitment by the initiator. The use of commitments gives a *normative* characterization to coordination [13,29], and allows reasoning about the agents' behavior [17]. The enactment entails the acceptance of the norms encoded via commitments, thus, our proposal realizes also the topmost layer of Figure 1, *Society*, while JaCaMo stops at the *Functionalities* level.

The proposal implements the desiderata for a Multi-agent System layered architecture in [16]. There, in order to better deal with cross-organizational business processes, the authors support a middleware giving the agents the possibility to reason directly on commitments. We do this with Jason agents: a Jason agent which enacted a protocol role is automatically notified, in its belief base, of the changes occurred in the social state. As a result, the agent can plan its behavior based also on this information. The artifact can also play the role of a monitor of the on-going interaction. This can be done because the only way for changing the social state is by way of the social actions. The monitoring functionality aims at verifying that the social commitments are satisfied. At the end of the interaction, those commitments that remain unsatisfied raise an exception. As such, the work sets along the line of [1] with the advantage that the use of commitments instead of global session types preserves the autonomy of the agents.

We believe this proposal to be an interesting basis for future developments; in particular, for studies aimed at identifying effective and efficient methodologies for engineering STS. The approach can also support extensions to richer norm expressions, that may for instance account for more complex conditions inside commitments (see [21]) or for richer languages where other kinds of constraints can be specified (see [4]). Finally, another interesting future development concerns the introduction of an agent typing system, along the lines of [3].

Acknowledgements

We thank the anonimous reviewers for their comments which were extremely helpful for improving the paper.

References

1. Davide Ancona, Sophia Drossopoulou, and Viviana Mascardi. Automatic generation of self-monitoring mass from multiparty global session types in jason. In *DALT*, volume 7784 of *LNCS*, pages 76–95. Springer, 2012.
2. Matteo Baldoni, Cristina Baroglio, and Federico Capuzzimati. 2COMM: a commitment-based MAS architecture. In *Post-Proc. of EMAS 2013, Revised Selected and Invited Papers*, number 8245 in LNAI, pages 38–57. Springer, 2013.
3. Matteo Baldoni, Cristina Baroglio, and Federico Capuzzimati. Typing Multi-Agent Systems via Commitments. In *Proc. of the 2nd Int. Workshop on Eng. MAS, EMAS*, 2014.
4. 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.
5. Matteo Baldoni, Cristina Baroglio, Viviana Mascardi, Andrea Omicini, and Paolo Torroni. Agents, multi-agent systems and declarative programming: What, when, where, why, who, how? In *25 Years GULP*, volume 6125 of *Lecture Notes in Computer Science*, pages 204–230. Springer, 2010.
6. Fabio L. Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. JADE - A Java Agent Development Framework. In *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 125–147. Springer, 2005.

7. Fabio L. Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE.* John Wiley & Sons, 2007.
8. 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.
9. Rafael H. Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah-Seghrouchni, Jorge J. Gómez-Sanz, João Leite, Gregory M. P. O'Hare, Alexander Pokahr, and Alessandro Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, 30(1):33–44, 2006.
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. Paolo Bresciani and Paolo Donzelli. A practical agent-based approach to requirements engineering for socio-technical systems. In *AOIS*, volume 3030 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2003.
13. Cristiano Castelfranchi. Principles of Individual Social Action. In *Contemporary action theory: Social action*, volume 2, pages 163–192, Dordrecht, 1997. Kluwer.
14. Albert Cherns. Principles of Socio-Technical Design. *Human Relations*, 2:783–792, 1976.
15. Amit K. Chopra. *Commitment Alignment: Semantics, Patterns, and Decision Procedures for Distributed Computing.* PhD thesis, North Carolina State University, Raleigh, NC, 2009.
16. Amit K. Chopra and Munindar P. Singh. An Architecture for Multiagent Systems: An Approach Based on Commitments. In *Proc. of ProMAS*, 2009.
17. Rosaria Conte, Cristiano Castelfranchi, and Frank Dignum. Autonomous Norm Acceptance. In *ATAL*, volume 1555 of *LNCS*, pages 99–112. Springer, 1998.
18. Fabiano Dalpiaz, Amit K. Chopra, and Soo Ling Lim. The 1st int. workshop on requirements engineering for social computing. In *RESC*, page 1. IEEE, 2011.
19. 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.
20. Michael Fisher, Rafael H. Bordini, Benjamin Hirsch, and Paolo Torroni. Computational logics and agents: A road map of current technologies and future trends. *Computational Intelligence*, 23(1):61–91, 2007.
21. Elisa Marengo, Matteo Baldoni, Cristina Baroglio, Amit K. Chopra, Viviana Patti, and Munindar P. Singh. Commitments with regulations: reasoning about safety and control in regula. In *AAMAS*, pages 467–474. IFAAMAS, 2011.
22. Viviana Mascardi, Maurizio Martelli, and Leon Sterling. Logic-based specification languages for intelligent software agents. *TPLP*, 4(4):429–494, 2004.
23. Felipe Meneguzzi, Pankaj R. Telang, and Munindar P. Singh. A first-order formalization of commitments and goals for planning. In *AAAI.* AAAI Press, 2013.
24. Felipe Rech Meneguzzi and Michael Luck. Norm-based behaviour modification in bdi agents. In *AAMAS (1)*, pages 177–184. IFAAMAS, 2009.
25. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.
26. Andrea Omicini and Franco Zambonelli. TuCSoN: a coordination model for mobile information agents. In *1st Int. WS on Innovative Internet Information Systems (IIIS'98)*, pages 177–187. IDI – NTNU, Trondheim (Norway), 8–9 June 1998.

27. Davide Porello, Davide Setti, Roberta Ferrario, and Marco Cristani. Multiagent Socio-Technical Systems. An Ontological Approach. In *Proc. of the 15th Int. Workshop COIN*, pages 1–15, 2013.
28. Alessandro Ricci and Andrea Santi. Cartago by example. http://www.emse.fr/ boissier/enseignement/-maop13/courses/cartagoByExamples.pdf.
29. Munindar P. Singh. An ontology for commitments in multiagent systems. *Artif. Intell. Law*, 7(1):97–113, 1999.
30. Munindar P. Singh. A social semantics for agent communication languages. In *Issues in Agent Communication*, volume 1916 of *LNCS*, pages 31–45. Springer, 2000.
31. Munindar P. Singh. Norms as a basis for governing sociotechnical systems. *ACM Trans. on Int. Sys. and Tech. (TIST)*, 5(1):22:1–22:21, December 2013.
32. Ian Sommerville. *Software Engineering*. Addison-Wesley, 9 edition, 2010.
33. 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.
34. Eric Trist. *The Evolution of Socio-technical Systems: A Conceptual Framework and an Action Research Program*. 1981.
35. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007.
36. Brian Whitworth and Adnan Ahmad. *Socio-Technical System Design*. The Interaction Design Foundation, Aarhus, Denmark, 2013.
37. P. Yolum and M. P. Singh. Commitment Machines. In *Intelligent Agents VIII, 8th Int. WS, ATAL 2001*, volume 2333 of *LNCS*, pages 235–247. Springer, 2002.