Social Computing with JaCaMo+2COMM4JASON

Social Computing with 2COMM4JASON

Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio

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

Abstract

Social Computing (SC) requires agents to reason seamlessly both on their social relationships and on their goals, beliefs. We claim the need to explicitly represent the social state and social relationships as resources, available to agents. We built a framework, based on JaCaMo, where this vision is realized and SC is implemented through social commitments and commitment protocols.

1998 ACM Subject Classification I.2.11 Artificial Intelligence, Distributed Artificial Intelligence, Coherence & co-ordination; multiagent systems.

Keywords and phrases Social Computing, Agent Programming, Commitments and Goals, Agents & Artifacts, JaCaMo

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 PROPOSAL AND MOTIVATION

Many systems, developed to support human users, require a transition from an individualistic to a societal perspective. For instance, Socio-Technical Systems (STS) are large-scale, multi-party, cross-organizational systems, which help stakeholders to interact and to use shared resources [11]. Such systems perform a *social computation* which is the sum of the independent contributions of autonomous, and heterogeneous, parties [10]. Traditional approaches to software engineering do not fit the needs of such systems, because they do not help capturing the social aspects of the computation, like the social relationships between the parties. The way suggested by STS is to foresee a specific layer that contains the regulations that norm the system behavior. This direction is followed by normative MAS, e.g. [7], which enrich MASs by representing the norms that rule the system. However, such approaches lack proper abstractions for capturing the *social state*, i.e. the set of relationships and dependencies that are created and exist along the course of events, which are the foundations on which the social behavior of the parties is established. Social relationships connect the interacting parties, they have a normative value (in that they allow agents to have expectations on one another), and they can be verified based just on the observable behavior of the agents. From a Software Engineering perspective, the advantage of explicitly representing the social state is to allow the realization of systems with a high degree of decoupling and of modularity of their components, avoiding to "hard code" the logic of interaction inside the code of the agents, whose executions are kept aligned by the social state itself.

Most of Multi-Agent frameworks and platforms do not explicitly account for the social state. We propose an agent framework, 2COMM4JASON, that, instead, does so by explicitly representing the social state through the social relationships and the rules that cause it to evolve along the interaction. Agents and social relationships are 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. On the other hand, existing social relationships affect the decisions and the behaviors of the agents they involve.

Our proposal exploits the Agents&Artifacts (A&A) meta-model [8] and reifies the social state as a set of *resources*, in a way that allows agents to seamlessly reason on them and on their beliefs, goals, etc. Thus parties can dynamically recognize, accept, refuse rules and relationships, as advised in [4]. Moreover, the social state, as a resource, can be used to monitor the interaction. Social relationships are modeled as commitments and the rules that cause the social state to evolve are modeled as commitment-based interaction protocols. The framework builds upon the JaCaMo platform [2], and the Jason language is extended so to allow reasoning on commitments.

2 2COMM4JASON

We model social relationships as *commitments* [9]: $C(x, y, r, p)$ 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 are conjunctions or disjunctions of events and commitments. Debtors are expected to satisfy the engagements they have taken – they are expected to behave so as to achieve the respective consequent conditions. Commitments satisfy the requirement that [5] in a system made of autonomous and heterogeneous actors, social relationships cannot but concern the observable behavior. They also satisfy the requirement [4] of having a normative value, consequently providing social expectations on the agents' behavior. As a consequence, they can be used by agents in their *practical reasoning* together with beliefs, intentions, and goals. A *commitment-based interaction protocol* is a collection of actions, whose social effects are expressed in terms of standard commitment operations (*create*, *cancel*, *release*, *discharge*, *assign*, *delegate*).

The framework is implemented based on JaCaMo [2], a platform integrating Jason (as an agent programming language), CArtAgO (as a realization of A&A), and Moise (as a support for realizing organizations). In JaCaMo a MAS is a Moise agent organization, which involves a set of Jason agents, all working in shared distributed artifact-based environments, programmed in CArtAgO. Environments are programmed as a dynamic set of artifacts, possibly distributed among various nodes of a network, that are collected into workspaces. They can be joined by agents at run-time and there agents can create, use, share artifacts to support their activities. Artifacts are computational, programmable system resources, that can be manipulated by agents. 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 [3] implements in Java, and extends, the agent programming language AgentSpeak(L). Jason agents have a BDI architecture: each has an own belief base, a set of ground (first-order) atomic formulas, and a set of plans (plan library). It is possible to specify achievement ('!') and test goals ('?'). 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. In JaCaMo, agent beliefs can also change due to the automatic propagation of the effects of actions executed in the environment. Plans are activated by the creation/deletion of some belief or goal.

We introduce commitments in Jason as terms of form: *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. *Status* is a further parameter that we use to keep track of the commitment state (created, satisfied, violated, conditional, detached, expired, pending, terminated). We introduced a class of artifacts that reify social

states. The belief bases of the agents focusing on such an artifact are aligned to the social state of the ongoing interaction: any modification of the latter is propagated to the former by the artifact itself by exploiting proper observable properties, that are added to or removed from the artifact properties. The artifact is responsible for maintaining the social state up-to-date, depending on the actions executed.

A protocol action is implemented as an artifact operation; its execution causes the update of the social state. An agent can execute a protocol action if its role matches with the one to which the action is associated. The check is transparent to the agent.

```
public class Cnp extends ProtocolArtifact {
  @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));
  } ...
```

Agent plans can be triggered by events involving commitments. 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, after a modification of the social state. For example, the plan $+cc(d, c, ant, cons, s) : \langle context \rangle \leftarrow \langle body \rangle$ is triggered when the commitment appears in the social state with the specified status. The plan may aim at achieving a change of the status of the commitment (e.g. the debtor will satisfy the consequent, the creditor will satisfy the antecedent and detach the commitment) or it may allow the agent to react to the event (e.g. collecting information). Similarly for commitment deletion. Commitments can also be used in contexts and in plans as test goals ($?cc(\ldots)$) or achievement goals ($!cc(\ldots)$). Addition or deletion of such goals can, as well, be managed by plans. For example, the plan $+!cc(d, c, ant, cons, s) : \langle context \rangle \leftarrow \langle body \rangle$ is triggered when the agent creates an achievement goal concerning a commitment. Consequently, the agent will act upon the artifact to create the desired social relationship. After the execution $cc(d, c, ant, cons, s)$ will hold in the social state and will be projected onto the belief bases of its parties.

This is an excerpt of Jason agent code for playing the role Initiator of the Contract Net Protocol.

```
+!startCNP : true
  <- makeArtifact("cnp","cnp.Cnp",[],C);
     focus(C); enact("initiator").
+enacted(Id,"initiator",R_Id) : true
  <- +enactment_id(R_Id);
     !cc(R_Id, "part", "propose",
           "(accept OR reject)","CONDITIONAL").
+!cc(My_R_Id, "part", "propose",
           "(accept OR reject)","CONDITIONAL")
  <- cfp("task-one").
+cc(My_R_Id, "part", "true",
           "(accept OR reject)", "DETACHED")
  : enactment_id(My_R_Id)
  <- !cc(My_R_Id, "part", "true",
           "(accept OR reject)", "SATISFIED").
+!cc(My_R_Id, "part", "true",
           "(accept OR reject)", "SATISFIED")
  : not evaluated
  <- +evaluated; ... find winner ...
     accept(Winner_R_Id).
     ... action 'reject' for all other proposals ...
+cc(Participant_R_Id, My_R_Id, "true",
           "(done OR failure)", "DISCHARGED")
```

```
24   :  done(Result) <- // collect results
25 +cc(Participant_R_Id, My_R_Id, "true",
26              "(done OR failure)", "DISCHARGED")
27   :  failure(Part_R_Id) <- true.
```

The agent playing this role creates the artifact that will be used for the interaction. The initiator agent can, then, execute *cfp*. When enough proposals will be received, the initiator agent evaluates them and decides which to *accept* and to *reject*. 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.

Agent behaviour is defined based on the existing social relationships and not on the process by which they are created. For instance, 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. Indeed, the framework provides a strong *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: the protocol is not implemented inside the agent code but it is a separate resource. Protocols can be updated separately from agents; as a consequence, the system maintainability is increased and the autonomy of the agents preserved. However, when an agent uses a protocol artifact, it accepts the commitments that may involve it and the rules the artifact reifies. This allows the interacting parties to perform practical reasoning based on expectations. Moreover, the artifact can act as a monitor of the interaction because this occurs through its roles, and detect violations that it can ascribe to the violators without agent introspection. Instead, in solutions that hard code the interaction rules, the check necessarily requires agent introspection.

We mean to extend to richer norm expressions, e.g. to account for temporal constraints [6, 1].

References

1 M. Baldoni, C. Baroglio, E. Marengo, and V. Patti, 'Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach', *ACM TIST*, **4**(2), 22:1–22:25, (March 2013).

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

3 R. H. Bordini, J. Fred Hübner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*, John Wiley & Sons, 2007.

4 R. Conte, C. Castelfranchi, and F. Dignum, 'Autonomous Norm Acceptance', in *ATAL*, volume 1555 of *LNCS*, pp. 99–112. Springer, (1998).

5 M. Dastani, D. Grossi, J.-J. Ch. Meyer, and N. A. M. Tinnemeier, 'Normative Multi-agent Programs and Their Logics', in *KRAMAS*, volume 5605 of *LNCS*, pp. 16–31. Springer, (2008).

6 E. Marengo, M. Baldoni, C. Baroglio, A. K. Chopra, V. Patti, and M. P. Singh, 'Commitments with regulations: reasoning about safety and control in REGULA', in *AAMAS*, pp. 467–474. IFAAMAS, (2011).

7 F. R. Meneguzzi and M. Luck, 'Norm-based behaviour modification in BDI agents.', in *AAMAS (1)*, pp. 177–184. IFAAMAS, (2009).

8 A. Omicini, A. Ricci, and M. Viroli, 'Artifacts in the A&A meta-model for multi-agent systems', *J. AAMAS*, **17**(3), 432–456, (2008).

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

10 M. P. Singh. Social computing: Principles, methods, and technologies, 2014. Invited talk at the First Int. Workshop on Multiagent Foundations of Social Computing.

11 I. Sommerville, *Software Engineering*, Addison-Wesley, 9 edn., 2010.