A Commitment-based Infrastructure for Programming Socio-Technical Systems

Matteo Baldoni; Cristina Baroglio; Federico Capuzzimati. A Commitment-based Infrastructure for Programming Socio-Technical Systems. ACM TRANSACTIONS ON INTERNET TECHNOLOGY. 14 (4) pp: 23:1-23:23.
DOI: 10.1145/2677206

A Commitment-based Infrastructure for Programming Socio-Technical Systems

MATTEO BALDONI, Università degli Studi di Torino, Dipartimento di Informatica
CRISTINA BAROGLIO, Università degli Studi di Torino, Dipartimento di Informatica
FEDERICO CAPUZZIMATI, Università degli Studi di Torino, Dipartimento di Informatica

Socio-Technical Systems demand an evolution of computing into social computing, with a transition from an individualistic to a societal view. As such, they seem particularly suitable to realize multi-party, cross-organizational systems. Multi-Agent Systems are a natural candidate to realize Socio-Technical Systems. However, while Socio-Technical Systems envisage an explicit layer that contains the regulations that all parties must respect in their interaction, and thus preserve the agents' autonomy, current frameworks and platforms require to hard-code the coordination requirements inside the agents. We propose to explicitly represent the missing layer of Socio-Technical Systems in terms of social relationships among the involved parties, i.e. in terms of a set of normatively defined relationships among two or more parties, subject to social control by monitoring the observable behaviour. In our proposal, social relationships are resources, available to agents, who use them in their practical reasoning. Both agents and social relationships are first-class entities of the model. The work also describes 2COMM4JADE, a framework that realizes the proposal by extending the well-known JADE and CArtAgO. The impact of the approach on programming is explained both conceptually and with the help of an example.

Categories and Subject Descriptors: I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; I.2.11 [**Distributed Artificial Intelligence**]: Languages and structures

General Terms: Design, Infrastructures

Additional Key Words and Phrases: Interaction protocols, commitments, social relations, middleware, artifacts, sociotechnical systems

ACM Reference Format:
Matteo Baldoni, Cristina Baroglio, and Federico Capuzzimati, 2014. Programming and Reasoning about Social Relationships: a Commitment-based Infrastrcture. *ACM Trans. Internet Technol.* 9, 4, Article 39 (March 2010), 23 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Modern, complex information technology systems require new software engineering techniques in order to scale adequately and to reduce risks of undesired, unpredictable behaviours [Sommerville et al. 2012]. In particular, traditional approaches do not fit the needs of large-scale, multi-party, cross-organizational systems, especially those needs which concern the design of interaction and distributed computation.

A different way of imagining and of engineering applications that fit the described setting is via *socio-technical systems* (STS for short) [Cherns 1976; Trist 1981]. These are systems where several, autonomous actors (people who use or interact with other

Author's addresses: M. Baldoni, C. Baroglio, F. Capuzzimati, Dipartimento di Informatica, Università degli Studi di Torino c.so Svizzera 185, I-10149 Torino (Italy).

DOI: http://dx.doi.org/10.1145/0000000.0000000

system components) use resources to achieve individual or shared goals. They 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 demand an evolution of computing [Whitworth and Ahmad 2013] into *social computing* [Dalpiaz et al. 2011], with a transition from an individualistic to a societal view, where notions like *social structure*, *role* and *norm* come into play. They can be conceived as a set of stacked layers [Sommerville et al. 2012], the lowest concerning physical resources, the others concerning higher and higher abstractions and functionalities (e.g. communication, application, business process). The topmost layer concerns the *laws* and the *regulations* that govern the operation of the system (the society). This level provides the *coordination mechanisms* that are necessary to the functioning of the system as well as the *expected observable behavior* of the system. As such, it supplies a base for the reasoning and for the deliberative activities of the participants and a tool for monitoring the correctness of the evolution of the system. In this way, a set of autonomous actors, which are decoupled by nature, can work and interact in a shared environment, realizing a form of self-governance [Singh 2014].

This work proposes a framework for realizing STS that builds upon *Multi-Agent Systems* (MAS) – a conceptual choice supported, among the others, by [Bresciani and Donzelli 2004; Porello et al. 2013; Singh 2014]. Agents represent stakeholders and act on their behalf, they are autonomous as far as the system allows them, always respecting the criteria and the preferences of the stakeholders they represent. Agent-oriented software engineers can choose from a substantial number of agent platforms [Mascardi et al. 2004; Bordini et al. 2006; Fisher et al. 2007; Baldoni et al. 2010]. The choice is related to very different and heterogeneous factors, like: scope and purpose of the system; formal model the platform is based on; richness of the agent programming language, if devised; platform support, maintenance and update; (graphical) tools for supporting design and development; simplicity of integration with other (agent) programming languages. Platforms and frameworks like JADE [Bellifemine et al. 2005], TuCSoN [Omicini and Zambonelli 1998], DESIRE [Brazier et al. 1997], JIAC [Thiele et al. 2009], all provide *coordination mechanisms* and *communication infrastructures* [Bordini et al. 2006]. We claim, however, that none of the current infrastructures for MAS is adequate to the purpose of realizing STS because none of them explicitly accounts for the topmost layer of STS, thus forcing the projection of the regulations directly inside the behaviors of the agents. As a consequence, MAS infrastructures do not really preserve autonomy and do not fit the high degree of decoupling which characterizes the participants of an STS. On the contrary, an explicit account of the topmost layer would clearly separate the agent specification from the coordination specification, not requiring any "hard-coding" of the regulations of the system inside the activities of the participants.

To overcome the limits of current MAS platforms, we propose to explicitly represent the topmost layer of STS in terms of *social relationships* among the involved participants, i.e. in terms of normatively defined relationships among two or more participants, that are subject to social control by monitoring their observable behavior. For supplying a normative characterization of social relationships, we propose to rely on *commitments*, which feature a social and observational semantics [Singh 2000], and in order to represent the coordination requirements we propose to rely on commitment-based protocols [Yolum and Singh 2002].

We also claim that the social relationships and commitment-based protocols should be integrated in the system in the form of *resources* because this allows participants to dynamically recognize, accept, manipulate, reason on them, and decide whether to conform to them. In particular, in a STS it is fundamental that the participants

explicitly accept the regulations, that are encode by them, because by this act they allow all the other participants, as well as the system, to have expectations on their behavior. This supplies a basis for coordination [Conte et al. 1999]. In other words, the topmost level of STS should be realized by way of first-class objects.

The framework that we propose, 2COMM4JADE, uses JADE as basic agent platform, which provides a FIPA compliant communication framework and an agent-developing middleware. In order to reify the social relationships we rely on the Agents & Artifacts meta-model (A&A) [Weyns et al. 2007; Omicini et al. 2008], which provides abstractions for environments and artifacts, that can be acted upon, observed, perceived, notified, and so on. When embodied inside artifacts, social relationships can be examined by the agents (to take decisions about their behavior), as advised in [Chopra and Singh 2009], used (which entails that agents accept the corresponding regulations), constructed, e.g., by negotiation, specialized, composed, and so forth. Last but not least, 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.

Summarizing, this work proposes to realize the society layer of STS as a set of social relationships, captured as commitments (Section 2). The normative characterization of the social relationships allows to decouple the programming of agents from the design of their interaction, and allows agents to reason on each other's expected behavior. Social relationships are actual resources, implemented through artifacts, that are made available to the agents, and are first-class entities of the model, as well as agents. The framework 2COMM4JADE (Section 3) realizes the proposal based on an extension of JADE and of CArtAgO. We show the impact of the proposal on programming by means of an example (Section 4).

2. MODELING SOCIAL RELATIONSHIPS: TOWARDS THE REALIZATION OF STS

In the 50's, the Tavistock Institute of Human Relations developed a new model of work organization, characterized by the assignment of responsibilities, the decentralization of work structures, the grouping of employees into mindful teams. This model is called *socio-technical design of socio-technical systems* [Trist 1981]. Different design principles were proposed as basis for socio-technical design. One of them is particularly relevant for the design of modern systems [Cherns 1976], the *Minimal Critical Specification*: only the essential must be specified. In his work the author, member of the Tavistock Institute, highlights how in the organization design the focus is traditionally put on *how* rather than on *what*. Quote: "In any case, it is a mistake to specify more than is needed because by doing so options are closed that could be kept open. *This premature closing of options is a pervasive fault in design.*".

In parallel with real-world work organization cases, socio-technical design principles were adapted to software and systems engineering [Sommerville et al. 2012]. The idea is to model complex software systems based on the *interactions* among their components, whose coordination (either collaborative or competitive) is the fundamental pillar. Besides interactions, norms and policies for resource usage are also provided. Inspire by [Sommerville et al. 2012], we see STS as structured into three architectural layers (Figure 1): *infrastructure*, *functionalities*, and *society*. The first concerns the (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 of the system.

The introduction of the agent metaphor, though having a different origin, represents a paradigm shift that moves along the lines of socio-technical design: from a *control-flow*-based organization of software (typical of the functional and the object-oriented paradigms) to a *message-based* one, shaped around message exchanges among

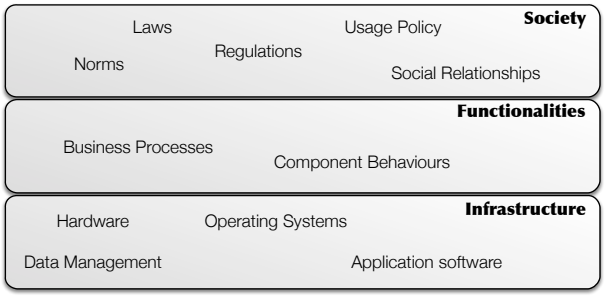

Fig. 1. Architectural macro-levels in a STS.

autonomous, computational entities. However, with reference to the architecture proposed in Figure 1, most of the existing Multi-Agent frameworks and platforms mix the third level with the second, rather than explicitly accounting for a social layer of the interaction and, thus, spread the coordination logic across the agents' implementations. Consider, for instance, the well-known JADE [Bellifemine et al. 2007] and Jason [Bordini et al. 2007], which respectively rely on the object-oriented and on the declarative paradigms, and consider the way they manage a simple interaction like the one captured by the FIPA Contract Net Protocol (CNP). In the JADE example implementation [Bellifemine et al. 2007, Sec. 5.4, page 100-107], agents execute one of two behaviours, 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 (*FSMBehaviour*). These behaviours are provided by the package *jade.proto*. In Jason [Bordini et al. 2007, Sec. 6.3, page 130], the interaction rules are again split into the behaviours of the interacting agents. Also in this case, and despite the declarative nature of Jason, each behaviour 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 layer of STS by representing social relationships among the agents. By social relationships we mean normatively defined relationships, and the expected patterns of interaction, between two or more agents, resulting from the enactment of *roles*, and subject to *social control*. 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' behaviour. It is, therefore, necessary to provide the agents the means to create and to manipulate, and to observe, to monitor, to reason and to deliberate on social relationships so to take proper decisions about their behaviour. We do so by exploiting *properly defined artifacts*, that reify both *interaction protocols*, defined in terms of social relationships, and the *sets of social relationships*, created during protocol execution. Such artifacts are made available to agents as resources.

The advantage of our proposal, compared to previously existing approaches, is that it clearly *decouples* the interaction design from the agent design, using artifacts to encode the coordination logic. Protocols provide a means of coordination, based on the *notification* of social events, e.g. the creation of a commitment. Interacting agents use artifacts to coordinate and interact, depending on the roles they play and on their objectives. One further advantage is that, thanks to artifacts and to commitments and to their normative value, it is possible to realize monitoring functionalities of the interaction, without imposing any specific behaviour. As such, our proposal respects the

Minimal Critical Specification principle advocated for STS, and preserves the agents' autonomy. Traditional approaches, on the contrary, do not support the decoupling of the design of agents and of the design of their interactions, which would be necessary in a cross-organizational setting: either the proper interactive capabilities are hard-coded in the implementation of the agents' behaviours (thus realizing no decoupling) or it is necessary to adopt off-the-shelf behaviours, which fully implement the specification but that are "injected" and that hide, in an external component, what should be the proactivity of the agent. This is, for instance, the case of JADE.

2.1. Social Relationships as Commitments

When modeling complex domains, with stakeholders belonging to different organizations or units of work, system engineers and analysts usually adopt the same formalism to describe behaviour of a particular component (a system resource and/or stakeholder) and interactions between components. BPMN is the preferred language; it allows to specify in detail what the expected behaviour of a resource (e.g. a printer, a web service, a web application) is, as well as that of an actor (e.g. the receptionist answering a call, the support team managing an help desk). It provides a graphical, intuitive means to organize tasks for achieving some result.

Problems arise when BPMN is used to design interactions between different stakeholders, each with its own set of business processes. Think, for example, to a health care system, whose purpose is to manage remote assistance to elder people and, if required, to send a medical support. Such system is a service integrator; it interrelates services provided by different entities when needed. Specifying how, for example, the physician will administer cures is out of its scope. The system only has to shape relationship between the physician and the patient who requested assistance. By using BPMN, one would build a multi-actor business process, where the tasks to be accomplished would be expressed in a procedural manner, and the interaction between the stakeholders (physician and patient) would be expressed as the sending and receiving of messages. This procedural view makes it difficult, for the involved actors, to be proactive or to adapt to variations in the environment, unless the business process is redesigned. This is basically due to the imposition of (often) unnecessary orderings. In other words, this approach does not suit the representation of business processes that are inherently *destructured*, as it is often the case in cross-organizational settings, and where seldom authority is centralized or clearly associated to a single stakeholder, but, rather, the involved actors are all peers that act in an autonomous way respecting an agreed regulation.

Pesic and van der Aalst [2006] proposed a shift of paradigm, passing from a procedural representation, leading to an over-constrained, non-realistic organization of work, to a declarative representation. In our opinion, this shift of paradigm satisfies the Minimal Critical Specification requirement illustrated before: specify "what" should occur rather than "how" making it occur. The question to answer, then, becomes: how to model interactions between autonomous entities without considering them as non-autonomous ones? We adopt the basic ideas used, among others, in [Telang and Singh 2011], where interactions and Cross-Organizational Business Processes are modeled in terms of regulated relations between entities, and where the founding element of a relation is a *commitment*.

Commitments define the agents normative context. In our proposal, they constitute the *top level of a socio-technical system*, the one which accounts for societal regulations [Sommerville 2010]. Commitments [Singh 1999] are represented with the notation $\mathsf{C}(x, y, r, p)$, capturing that the agent x commits to the agent y to bring about the consequent condition p when the antecedent condition r holds. Antecedent and consequent conditions generally are conjunctions or disjunctions of events and commit-

ments. When r equals $\top$, we use the short notation $\mathsf{C}(x, y, p)$ and the commitment is said to be *active*. 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.

The stakeholders share a *social state* that contains commitments. Stakeholders affect the social state while performing their activities. Specifically, the manipulation of commitments is done by means of the standard operations *create*, *cancel*, *release*, *discharge*, *assign*, *delegate*. As in [Singh 1999], 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. For details see [Singh 1999; Yolum and Singh 2002; Chopra 2009].

The next question to answer is how to integrate, inside the outlined agent and artifact setting, the rules that regulate the process by which social relationships are created and manipulated. In our proposal, this is done by relying on the notion of *commitment-based interaction protocol*. This kind of protocol consists of a set of actions (process activities), whose semantics is shared, and agreed upon, by all of the participants to the interaction [Yolum and Singh 2002; Chopra 2009]. The semantics of the social actions is given in terms of the operations which allow the modification of the social state and that have already been described.

Along the line of [Telang and Singh 2010], we propose to start protocol modeling by identifying the necessary social relationships, that are, then, represented as commitments. Only afterwards the focus shifts to identifying the activities that make them evolve during the interaction. This represents an inversion w.r.t. the traditional, activity-centric design phase: the evolution of commitments guides the definition of the activities, needed by the interacting parties. This is also a difference with the normal use of commitment protocols where commitments just give semantics to the protocol actions: here, commitments are the focus, they are first-class objects in the deliberative activity of the agents, and can be manipulated by means of the protocol actions.

From an organizational perspective, a protocol is structured into a set of *roles*. Roles and agents are different entities, and we assume that roles cannot live autonomously: they exist in the system in view of the interaction, because agents, for interacting, use artifacts and execute actions on them. We follow the ontological model for roles proposed in [Boella and van der Torre 2007], and brought inside the object-oriented paradigm in [Baldoni et al. 2007], which is characterized by three aspects: (1) *Foundation*: a role must always be associated with the institution it belongs to and with its player; (2) *Definitional dependence*: the definition of the role must be given inside the definition of the institution it belongs to; (3) *Institutional empowerment*: the actions defined for the role in the definition of the institution have access to the state of the institution and of the other roles, thus, they are called powers; instead, the actions that a player must offer for playing a role are called requirements.

2.2. Social Relationships as Resources

Following [Conte et al. 1999], we need social relationships to *be recognized* as having a normative force (so that they will provide social expectations on the stakeholders' behaviour), to *be accepted* explicitly by the participants to the interaction, to *be inspected* so as to allow stakeholders to decide whether conforming to them. To this aim, we propose to explicitly model social relationships as *resources*, that are made available to the interacting peers, in the very same way as other kinds of resources. Moreover, in a system made of autonomous and heterogeneous actors, social relationships cannot but concern the observable behaviour [Dastani et al. 2009].

Given that stakeholders and social relationships are both first-class entities, that interact in a bi-directional manner, how to model these two elements in the Multi-agent paradigm? This provides a single first-class entity, the *agent*, which is not suitable for representing inspectable elements because an agent's internal state is not available to external entities by definition. To solve this issue, we adopt the *Agents and Artifacts* (A&A) meta-model [Weyns et al. 2007; Omicini et al. 2008], that extends the agent paradigm with another primitive abstraction, the *artifact*. An artifact is a computational, programmable system resource, that can be manipulated by agents, residing at the same abstraction level of the agent abstraction class. The A&A paradigm provides ways for defining and organizing workspaces, i.e. logical groups of artifacts, that can be joined by agents at runtime and where agents can create, use, share and compose artifacts to support individual and collective, cooperative or antagonistic activities. The environment is itself programmable, and encapsulates services and functionalities. For their very nature, artifacts can encode a mediated, programmable and observable means of communication and coordination between agents. Following [Baldoni et al. 2011], we rely on artifacts to realize the topmost layer of STS.

We interpret the fact that an agent uses an artifact as the explicit acceptance, by the agent, of the regulation encoded by that artifact, and modeled by the interaction protocol that the artifact reifies. This is an important aspect 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; when this does not happen, a violation is raised. Relying on artifacts allows also other kinds of manipulation, including (but not limited to) an agent playing a role in the interaction, and an agent observing the interaction that is being carried on.

A role represents a way of manipulating the social state and belongs to the artifact which reifies its protocol. In other words, roles are supplied by protocols and, in turn, supply actions that enable interaction through the artifacts, that reify such protocols. They are *powers* for modifying the social state, e.g. by creating commitments. On the other hand, the agents that will be the *role players* need to satisfy the related *requirements*: specifically, in order to play a role an agent needs to have the capabilities of satisfying the related commitments – capabilities which can be internal of the agent or supplied as powers as well. For example, in order to play the role *Initiator* of the Contract Net Protocol (used later in the paper), an agent needs to be able to satisfy the commitment of assigning tasks to some of the participants.

Finally, in our proposal, the artifact maintains both the social state, with all the commitments, and further elements that contribute to the state of the interaction, and that are stored in a blackboard-style memory, that we call communication state. For instance, in the CNP case, the description of the task for which the call is issued, is recorded in this area. Agents, playing roles, can inspect the state of the interaction or be notified when events of interest occur.

Summary. We claim that an agent-based framework for realizing STS should satisfy the following requirements. First of all, there is the need of an explicit representation of the social relationships that are created and that evolve along the interaction. Such relationships should have a normative force: thus, they will allow agents to have expectations on the behavior of their parties. We decided to use commitments to represent social relationships and to rely on commitment-based protocols to represent the way in which social relationships evolve. Then, we believe that social relationships should be first-class objects, which can be used for programming the agent behavior. For this reason, we decided to make social relationships actual resources, made available to the agents through artifacts, and manipulable through interaction protocols.

3. 2COMM4JADE: A COMMITMENT-BASED INFRASTRUCTURE FOR SOCIAL RELATIONSHIPS

After providing the background, we describe the developed implementation framework, that we named 2COMM4JADE. The focus is to provide adequate support for programming social relationships by exploiting a declarative, interaction-centric approach and by relying on existing technologies as far as possible. Specifically, the implementation framework combines two well-known platforms: JADE [Bellifemine et al. 2005] and CArtAgO [Ricci et al. 2011]. JADE agents represent stakeholders, commitment-based interaction protocols are realized as CArtAgO artifacts.

JADE is a popular and industry-adopted agent framework. It offers to developers a Java middleware that is FIPA-compliant [Foundation for Intelligent Physical Agents 2002]. Its robustness and well-proven reliability makes JADE a preferred choice in developing MAS. A JADE-based system is composed of one or more *containers*, each grouping a set of agents in a logical *node* and representing a single JADE runtime. The overall set of containers is called a *platform*, and can spread across various physical hosts. The resulting architecture hides the underlying layer, allowing support for different low-level frameworks (JEE, JSE, JME, etc.). JADE provides communication and infrastructure services, allowing agents, that have been deployed in different containers, to discover and interact with each other.

CArtAgO is a framework based on the A&A meta-model [Weyns et al. 2007; Omicini et al. 2008]. It 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. CArtAgO provides an API to program *artifacts* that agents can use, regardless of the agent programming language or the agent framework used. This is possible by means of the *agent body* metaphor: CArtAgO provides a native agent entity, which allows using the framework as a complete MAS platform as well as it allows mapping the agents of some platform onto the CArtAgO agents, which, in this way, becomes a kind of "proxy" in the artifacts workspace. The former agent is the mind, that uses the CArtAgO agent as a body, interacting with artifacts and sensing the environment. An agent interacts with an artifact by means of public *operations*, which can be equipped with *guards*: conditions that must hold in order for operations to produce their effects. When guards do not hold, actions can still be performed but without consequences.

2COMM4JADE is organized as follows. JADE supplies standard agent services, i.e. message passing, distributed containers, naming and yellow pages services, agent mobility. When needed, an agent can enact a protocol role, which provides a set of operations by means of which agents participate in a mediated interaction session. Protocol roles are provided by *communication artifacts*, that are implemented by means of CArtAgO. Each communication artifact corresponds to a specific protocol enactment and maintains an own social state and an own communication state.

Figure 2 reports an excerpt of the 2COMM4JADE UML class diagram[1]. Let us get into the depths of the implementation:

— *CommunicationArtifact* (CA for short) provides the basic communication operations *in* and *out* for allowing mediated communication. by means of which agents respectively ask to play or to give up playing a role. CA extends an abstract version of the *TupleSpace* CArtAgO artifact: briefly, a blackboard that agents use as a tuple-based

[1]The source files of the system and examples are available at the URL http://di.unito.it/2COMM.

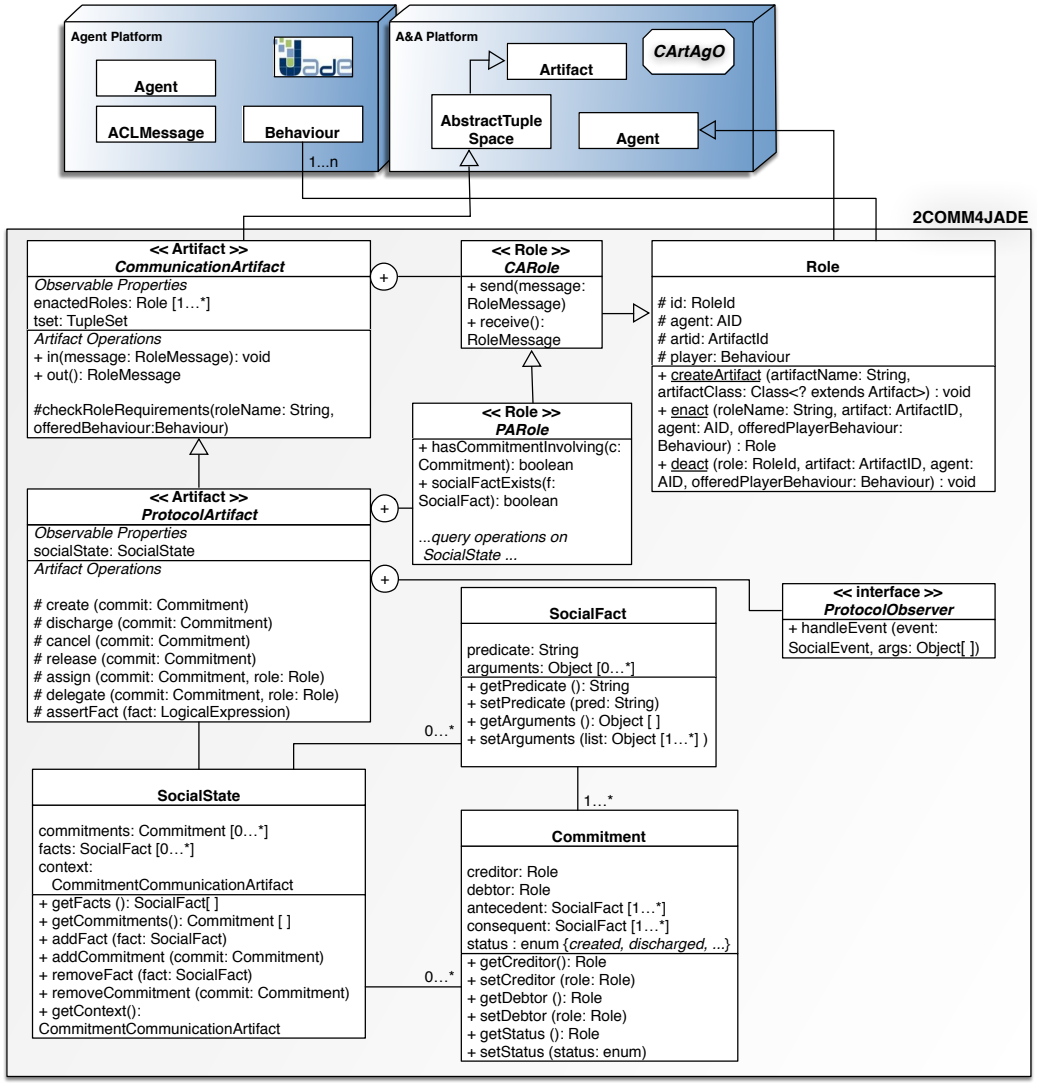

Fig. 2. Excerpt of the UML class diagram of 2COMM4JADE.

coordination means. In and out are, then, operations on the tuple space. CA also traces who is playing which role by using the property *enactedRoles*.

— Class *Role* extends the CArtAgO class *Agent*, and contains the basic manipulation logic of CArtAgO artifacts. Thus, any specific role, extending this super-type, will be able to perform operations on artifacts, whenever its player will decide to do so. Role provides static methods for creating artifacts and for *enacting*/*deacting* roles. This is done by passing a reference to the JADE agent behaviour that will actually play the role.

— The class *CARole* is an inner class of CA and extends the Role class. It provides the *send* and *receive* primitives, by which agents can exchange messages. Send and receive are implemented based on the *in* and *out* primitives provided by CA.
— *ProtocolArtifact* (PA for short) extends CA and allows modeling the social layer with the help of commitments. It maintains the state of the on-going 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 life-cycle and for the assertion/retraction of facts. Operations on commitments are realized as *internal operations*, that is, they are not invokable directly: the protocol social actions will use them as primitives to modify the social state. We refer to modifications occurred to the social state as *social events*. Being an extension of CA, PA maintains two levels of interaction: the social one (based on commitments), and the communication one (based on message exchange).
— The class *PARole* is an inner class of PA and extends the CARole class. It provides the primitives for *querying the social state*, e.g. for asking the commitments in which a certain agent is involved, and the primitives that allow an agent to become, through its role, an *observer of the events* occurring in the social state. For example, an agent can query the social state to verify if it contains a commitment with a specific condition as consequent, via the method `existsCommitmentWithConsequent(InteractionStateElement el)`. Alternatively, an agent can be notified about the occurrence of a social event, provided that it implements the inner interface *ProtocolObserver*. Afterwards, it can start observing the social state. PARole also inherits the communication primitives defined in CARole.

In order to specify a *commitment-based interaction protocol*, it is necessary to extend PA by defining 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. Since we want agents to act on artifacts only through their respective roles, when defining a protocol it is also necessary to create the roles. We do so by creating as many extensions of PARole as protocol roles. These extensions are realized as inner classes of the protocol: each such class will specify, as methods, the powers of a role. Powers allow agents who play roles to actually execute artifact operations. The typical schema will be:

```
1  public class MyProtocolArtifact extends ProtocolArtifact {
2    // ...
3    static {
4      addEnabledRole("Role1", Role1.class);
5      addEnabledRole("Role2", Role2.class);
6      // ...
7    }
8    // MY PROTOCOL ARTIFACT OPERATIONS
9    @OPERATION
10   public void op1(...) {
11     // prepare a message, if needed; in that case,
12     send(message);
13     // modify the social state, e.g. create commitment, update commitment
14   }
15   // ...
16   // INNER CLASSES for ROLES
17   public class Role1 extends PARole {
18     public Role1(Behaviour player, AID agent) {
19       super("Role1", player, agent);
20     }
21     // define social actions for Role1
22     public void action1(...) {
23       doAction(this.getArtifactId(), new Op("op1", ..., getRoleId()));
```

```
24      }
25      // ...
26    }
27    public class Role2 extends PARole {
28      // ...
29    }
30    // ...
31  }
```

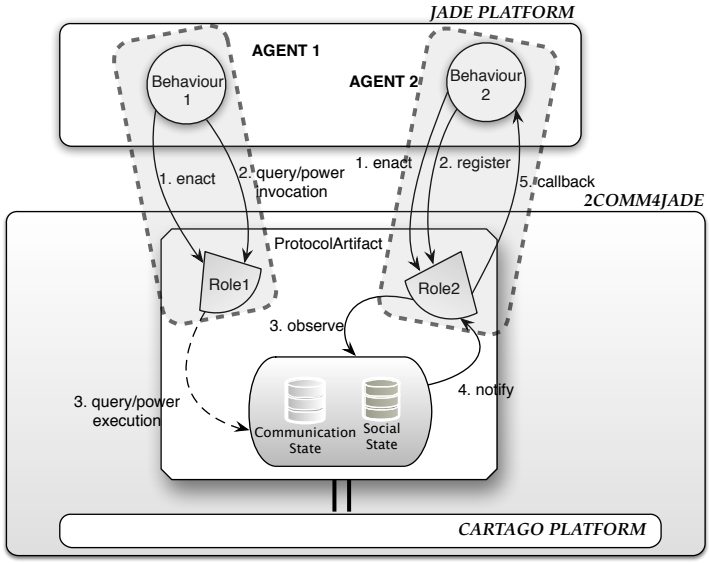

Fig. 3. 2COMM4JADE at runtime.

Let us, now, consider Figure 3, which sketches the relationships between the various components of the platform at run-time. When a JADE agent plays a role, its behaviour will use the powers offered by the role itself in order to act upon the social and communicative layers, offered by the artifact which implements the interaction protocol. In this context, the agent behaviour is extended with the powers of the role: the agent will be free to decide if and when using its new powers, based on its own business logic. The business logic is programmed by exploiting both knowledge which is internal to the agent –and that is represented and managed according to the agent model that is used–, and the social relations that are reified in the social state of the communication artifact. This can be done:

(1) either by exploiting the *query powers*, supplied by the role and inherited from PARole (Agent 1 in the figure);
(2) or by relying on a *social event-driven approach* (Agent 2 in the figure).

In case (1), the agent directly manages when probing the occurrence of social events and manage them. In case (2), the role registers as an observer of all events occurring in the social state (including all operations on commitments). When the interaction protocol notifies an event to the role, this calls back the behaviour of its player, passing the occurred event. The social event handler, defined in the behaviour, manages the event. Let us, now, describe a *methodological schema* for handling a social event.

ALGORITHM 1: Methodological Schema for Handling Social Events.

Data: Social event se notified to agent a
Data: Set S of social events of interest
Result: Add a behaviour to the behaviour set of a
Check whether se involves a commitment c which has a as debtor or creditor;
if $se \in S$ **then**
  b = behaviour that handles the occurred modification of c;
  add b to the set of behaviours of a;
end

The proposed algorithm explains how to use the notifications performed by the protocol artifact to the agent. Depending on which social event occurs, i.e. on which commitment is modified and how (e.g. added, discharged, detached), a specific behaviour is added and scheduled for execution, competing with others already present in the behaviour set. The agent designer can choose which social events are to be tackled by the agent. The following is the pseudo-code of an example implementation that agrees with the schema:

```
1  public class MyBehaviour extends SomeJadeBehaviour implements ProtocolObserver {
2    [ ... ]
3    public void action() {
4      ArtifactId art = Role.createArtifact(myArtifactName, MyArtifact.class);
5      myRole = (SomeRole) (Role.enact(MyArtifact.ROLE_NAME, art, this, myAgent.getAID()));
6      myRole.startObserving(this);
7      // add the initial behaviour of the agent
8    }
9    public void handleEvent(SocialEvent e, Object... args) {
10     if (e.getElementChanged().getElType() == interactionStateElementType.COMMITMENT) {
11       Commitment c = (Commitment)e.getElementChanged();
12       if (c.getLifeCycleStatus() == ... // e.g. LifeCycleState.DETACHED
13           && c.getDebtor().equals(...)  // some role, e.g. myRole.getRoleId()
14           && c.getCreditor().equals(...)// some role
15           && c.getConsequent().equals(...))) // a condition, e.g. "accept OR reject"
16       {
17         myAgent.addBehaviour(...);  // a behaviour to handle the case
18       } else if ( ... ) {
19         myAgent.addBehaviour(...);  // a behaviour to handle another case
20       } else
21         // ...                  // behaviours for different cases
22       }
23     }
24   }
25 }
```

The agent behaviour implements *ProtocolObserver*, thus allowing the agent to be notified about social events. This will be done after the agent executes the method *startObserving*, specifying which artifact it requires to observe. The implementation of *ProtocolObserver* requires, in turn, the implementation of the method *handleEvent*. Here, it is possible to test the kind of event that was notified (line 10), and in particular whether it concerns a commitment. If this is the case, it is possible to further check specific conditions of interest on the commitment, including its state, the identity of its debtor and/or creditor, the antecedent or consequent condition (lines 12-15). The agent will, then, add appropriate behaviours to handle the detected situation. The handler represents the classical agent *sense-plan-act cycle*, rephrased into “sense the social state”, “activate behaviors according to social events”, “schedule behavior execution”.

4. PROGRAMMING SOCIAL RELATIONSHIPS: AN EXAMPLE

FinancialMAS is an example application of 2COMM4JADE. Inspired by [European Parliament 2004], it allows financial interactions between three categories of stakeholders: (1) investors, (2) financial promoters and (3) banks. Investors have the goal

Table I. Responsibility Table for FinancialMAS.

Agent Type	No.	Responsibility
Investor agent (IA)	1.1	Let investor search for investments proposals
	1.2	Assist investor in setting search parameters and data
	1.3	Support the individuation of the investor's risk profile
	1.4	Support in proposal acceptance
	1.5	Withdraw from an investment contract
Financial Promoter agent (FP)	2.1	Respond to investment searches
	2.2	Assist financial promoter in risk-classifying financial products
	2.3	Determine the investor's profile
	2.4	Support individuation of the investor's risk profile
Bank agent (BA)	3.1	Support bank in investment contract subscription
	3.2	Assist bank in investment conclusion
Financial Provider agent (FV)	4.1	Provide financial and aggregate news information
Integration agent (IntA)	5.1	Serve and support integration with legacy bank informative systems

to place investments; promoters to propose and finalize investment contracts, on behalf of a bank and on the basis of current investor's profile; banks to finalize contracts placed by promoters.

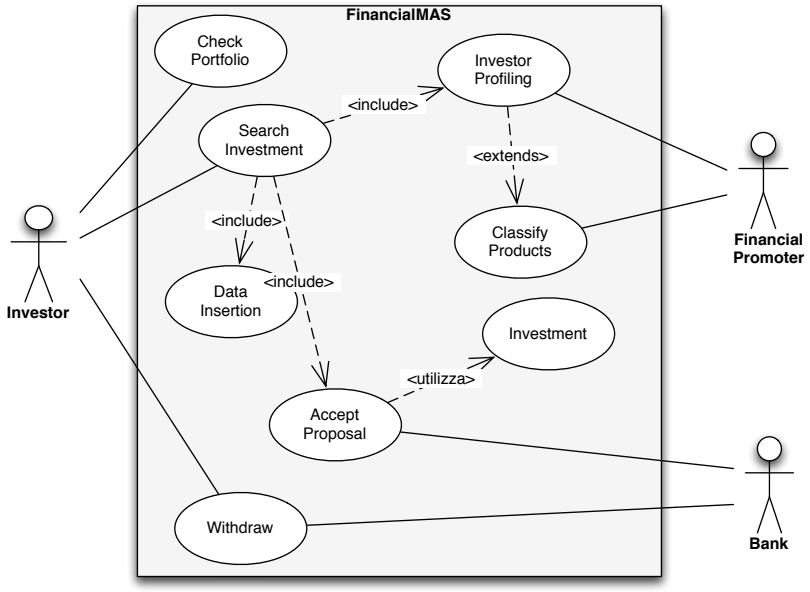

Fig. 4. The FinancialMAS Use Cases.

Figure 4 reports a part of the UseCase Diagram of the system, pointing out the macro-functionalities the system should offer and the involved stakeholders. Starting from this, we individuated a collection of agent types and environment artifacts. Each agent has *responsabilities*, in the specific case, tasks it should accomplish (see Table I), and related *social relationships*. Table II reports the *Interaction Table* for the domain. It contains the "patterns of interactions" that allow performing the various tasks, and that we realize as interaction protocols. Specifically, we rely on standard FIPA protocols and implement them as *protocol artifacts*.

Table II. Interaction Table for FinancialMAS: who interacts with whom, to fulfill which duty, by using which protocol.

Interaction	R.ty	Interaction Protocol	Role	With	When
Investor Agent					
Search Investment	1.1	CNP	Initiator	FP	Investor searches an investment
Profiling	1.3	Query	Participant	FP	Investor chose a Financial Promoter
Proposal Acceptance	1.4	Query	Participant	BA	Investor chose a financial product
Withdraw	1.5	Request	Initiator	BA	After Investor accepted a proposal
Financial Promoter Agent					
Respond to Search	2.1	CNP	Participant	IA	Investor searches an investment
Profiling	2.3	Query	Initiator	IA	Investor chose a Financial Promoter
Fin. Prod. Classif.	2.2	Query	Initiator	FV	FP starts fin. prod. classif.
Bank Agent					
Proposal Acceptance	3.1	Query	Initiator	IA	Investor chose a financial product
Withdraw	3.3	Request	Participant	IA	After Investor accepted a proposal
Financial Provider Agent					
Fin. Prod. Classif.	4.1	Query	Participant	FP	FP starts fin. prod. classif.

Let us focus on the Contract Net Protocol (CNP for short) [Foundation for Intelligent Physical Agents 2002], which is used inside FinancialMAS to manage the interaction between the Investor and the Financial Promoter when the former looks for some investment (Table II). We adopt the following CNP formulation:

cfp **causes** create(*C(i, p, propose, accept* $\vee$ *reject)*)
accept **causes** *none*
reject **causes** release(*C(p, i, accept, done* $\vee$ *failure)*)
propose **causes** create(*C(p, i, accept, done* $\vee$ *failure)*)
refuse **causes** release(*C(i, p, propose, accept* $\vee$ *reject)*)
done **causes** *none*
failure **causes** *none*

where i stands for the role *Initiator* and p for *Participant*. The instance used by FinancialMAS will tie the Investor to the i and the Financial Promoter to p. Initiator supplies its player the powers *cfp* (call for proposal), *accept*, and *reject*. The first allows the initiator to ask participants for proposals for solving a task of interest. If a proposal is chosen, action *accept* notifies the winner and all other proposals are rejected. The role participant supplies its player the powers *propose*, *refuse*, *done*, and *failure*. Action *propose* allows a participant to supply a solution for a task, action *refuse* allows declining the invitation to send a proposal. If a proposal is accepted, the winning participant is expected to execute the task and either provide the result by means of the action *done* or communicate its failure. Powers allow agents to affect the social state. For instance, when an agent playing the role Initiator executes *cfp*, the social state is modified by creating the commitment *C(i, p, propose, accept* $\vee$ *reject)*. This addition binds i to either accept or reject a proposal, if one is received. The agent is free to decide not only which course of action to take but also how to realize acceptance or rejection.

Let us see how CNP is realized in 2COMM4JADE. The class CNP extends ProtocolArtifact and implements as artifact operations all the protocol actions. In the sketch below we detail, for the sake of brevity, only the action *cfp*. Lines 6–10 represent the construction and sending of a call for proposals, which will be inserted in the blackboard and made available to participants. The recipient of the message is left generic because the Initiator may not know its peers, who may join even later in the future. Lines 12–15 modify the social state by adding commitments and by stating that the

event *cfp* has occurred. Lines 21–27 defines the role in terms of its powers. Here (lines 23–25), in fact, the power *cfp* is defined in terms of the homonym artifact operation.

```
1  public class CNP extends ProtocolArtifact {
2    // ...
3    @OPERATION
4    public void cfp(Task task, RoleId initiator) {
5      // send message with the task
6      RoleMessage cfp = new RoleMessage();
7      RoleId dest = new RoleId(PARTICIPANT_ROLE, RoleId.GENERIC_ROLE);
8      cfp.setContent(task);  cfp.setRoleSender(initiator);
9      cfp.setRoleReceiver(dest);  cfp.setPerformative(ACLMessage.CFP);
10     send(cfp);
11     // update the social state
12     createAllCommitments(new Commitment(initiator, dest, "propose",
13       new CompositeExpression(LogicalOperatorType.OR,
14         new Fact("accept"), new Fact("reject"))));
15     assertFact(new Fact("cfp", initiator, task));
16   }
17   @OPERATION
18   public void propose(Proposal prop, RoleId participant, RoleId initiator) {
19     // ...
20   }
21   // ...
22   // Role classes
23   public class Initiator extends PARole {
24     // ...
25     public void cfp(Task task) {
26       doAction(this.getArtifactId(), new Op("cfp", task, getRoleId()));
27     }
28     // ...
29   }
30   public class Participant extends PARole {
31     public void propose(Proposal proposal, RoleId proposalSender) {
32       // ...
33     }
34     // ...
35   }
36 }
```

The following, instead, is a sketch of the definition of a JADE agent, playing the role Initiator by using 2COMM4JADE. In this example, the artifact is created by the agent which, then, enacts the role of interest. At line 8, the agent uses the power *cfp* through the role it plays. Finally, at line 10, it inspects if someone committed to accomplish the task, and reads the proposals in the tuple space in order to make a choice.

```
1  public class InitiatorAgent extends Agent {
2    // ...
3    public class InitiatorBehaviour extends OneShotBehaviour {
4      // ...
5      public void action() {
6        ArtifactId art = Role.createArtifact(ARTIFACT_NAME, CNPArtifact.class);
7        initiator = (Initiator)(Role.enact(CNPArtifact.INITIATOR_ROLE, art, this,
8          myAgent.getAID()));
9        initiator.cfp(t);
10       // ...
11       boolean proposalPerformed = initiator.existsCommitmentWithConsequent(
12         new CompositeExpression(LogicalOperatorType.OR,
13           new Fact("done"), new Fact("failure"));
14       // ...
15     }
16   }
17 }
```

More interesting is to exploit the feature of 2COMM4JADE, which allows roles to register for the notification of social events. Thanks to the event handler, modifications to the social state will add specific behaviours to the agents, aimed at tackling the case. The code is organized by following the schema described in Algorithm 1. Figure 5 contains the UML diagram for this solution.

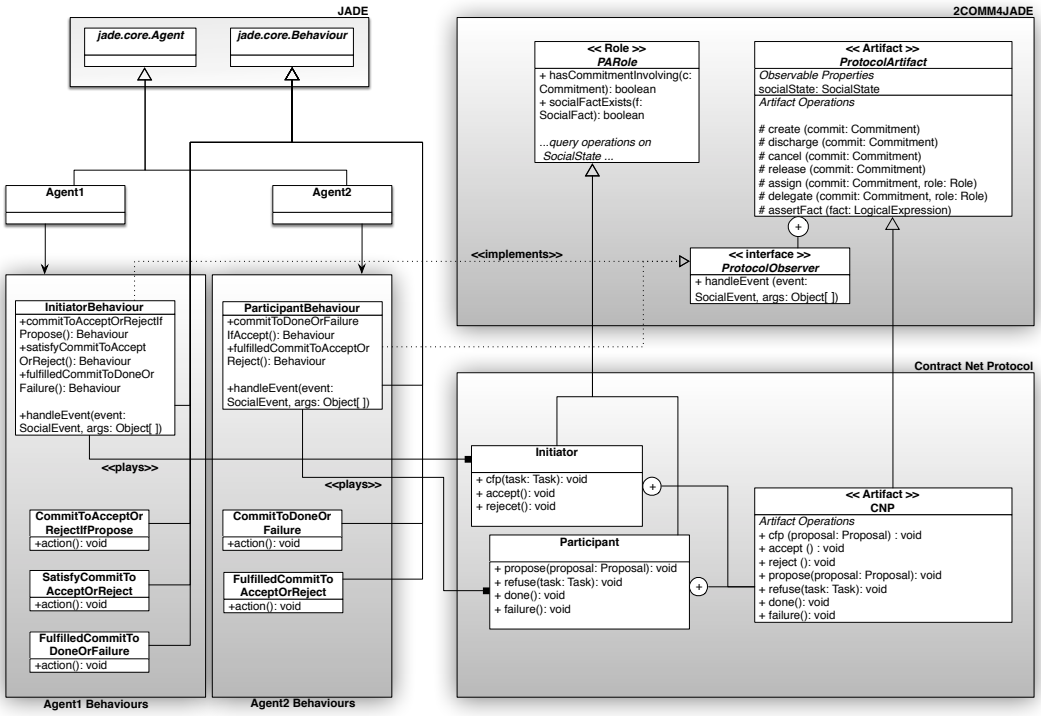

Fig. 5. CNP with 2COMM4JADE and support for event handling.

```
1  public abstract class InitiatorBehaviour extends OneShotBehaviour
2        implements ProtocolObserver {
3    public String artifactName;
4    protected Initiator initiator;
5    public abstract Behaviour commitToAcceptOrRejectIfPropose();
6    public abstract Behaviour satisfyCommitToAcceptOrReject();
7    public abstract Behaviour fulfilledCommitToDoneOrFailure();
8    public InitiatorBehaviour(String artifactName){
9      this.artifactName = artifactName;
10   }
11   public void action() {
12     ArtifactId art = Role.createArtifact(artifactName, CNPArtifact.class);
13     initiator = (Initiator) (Role.enact(CNPArtifact.INITIATOR_ROLE, art, this,
14       myAgent.getAID()));
15     initiator.startObserving(this);
16     myAgent.addBehaviour(this.commitToAcceptOrRejectIfPropose());
17   }
18   public void handleEvent(SocialEvent e, Object... args) {
19     if (e.getElementChanged().getElType() ==
20               interactionStateElementType.COMMITMENT) {
21       Commitment c = (Commitment)e.getElementChanged();
22       if (c.getLifeCycleStatus() == LifeCycleState.DETACHED
23               && c.getDebtor().equals(initiator.getRoleId())
24               && c.getCreditor().equals(new RoleId(
25                CNPArtifact.PARTICIPANT_ROLE, RoleId.GROUP_ROLE))
26               && c.getConsequent().equals("accept OR reject"))) {
27         myAgent.addBehaviour(satisfyCommitToAcceptOrReject());
28       } else {
29         myAgent.addBehaviour(fulfilledCommitToDoneOrFailure());
30       }
31     }
```

```
32   }
33 }
```

After line 14, all events occurring in the social state are notified to the role Initiator, which will handle them by executing *handleEvent* after a callback. The above abstract behaviour is extended by the concrete behaviour of the agent that plays the role Initiator. In particular, here we find the methods that create the actual behaviours for managing the social events.

```
1  public class InitiatorAgent extends Agent {
2    // ...
3    public class InitiatorBehaviourImpl extends InitiatorBehaviour {
4      public static final String ARTIFACT_NAME = "CNP-1";
5      public InitiatorBehaviourImpl() {
6        super(ARTIFACT_NAME);
7      }
8      public Behaviour commitToAcceptOrRejectIfPropose() {
9            return new CommitToAcceptOrRejectIfPropose(initiator);
10     }
11     public Behaviour satisfyCommitToAcceptOrReject() {
12       return new SatisfyCommitToAcceptOrReject(initiator);
13     }
14     public Behaviour fulfilledCommitToDoneOrFailure() {
15       return new FulfilledCommitToDoneOrFailure(initiator);
16     }
17   }
18 }
```

As an example, we describe the behaviour *SatisfyCommitToAcceptOrReject*, which gathers proposals and selects the one to accept.

```
1  public class SatisfyCommitToAcceptOrReject extends OneShotBehaviour {
2    Initiator initiator = null;
3    ArrayList<Proposal> proposals = new ArrayList<Proposal>();
4    public SatisfyCommitToAcceptOrReject(Initiator initiator) {
5      super();
6      this.initiator = initiator;
7    }
8    public void action() {
9      ArrayList<RoleMessage> propos = initiator.receiveAll(ACLMessage.PROPOSE);
10     for (RoleMessage p : propos) {
11       proposals.add((Proposal) (p.getContents()));
12     }
13     initiator.accept(proposals.get(0));
14     for (int i = 1; i < proposals.size(); i++) {
15       initiator.reject(proposals.get(i));
16     }
17   }
18 }
```

In 2COMM4JADE, an agent consists of a set of behaviours aimed at accomplishing given social relationships: such behaviours depend neither on when nor on how the social relationships of interest are created inside the social state. These aspects are, in fact, encoded in the protocol artifact that creates them based on the actions the agents perform. As a consequence, modifying how or when a social relationship is created does not have any impact on the agent implementation. We show this with the help of two examples where the same commitments (from the Initiator to the Participants, and concerning the acceptance/rejection of proposals) are detached respectively when a given number of proposals is received (Listing 1) or when a deadline expires (Listing 2).

Listing 1. Detach on number of proposals reached.

```
1  public class CNPArtifact extends ProtocolArtifact {
2    boolean acceptingProposals = true;
3    // ...
4    @OPERATION
5    public void propose(Proposal prop, RoleId participant, RoleId initiator) {
```

```
6       if (!acceptingProposals)
7         failed("No more proposals allowed.");
8       prop.setRoleId(participant);
9       RoleMessage proposal = new RoleMessage();
10      proposal.setContents(prop);
11      proposal.setRoleSender(participant);
12      proposal.setRoleReceiver(initiator);
13      proposal.setPerformative(ACLMessage.PROPOSE);
14      send(proposal);
15      assertFact(new Fact("propose", participant, prop));
16      createCommitment(new Commitment(participant, initiator, "accept", "done OR failure"));
17      if (actualProposals == numberMaxProposals) {
18        acceptingProposals = false;
19        RoleId groupParticipant = new RoleId(PARTICIPANT_ROLE, RoleId.GROUP_ROLE);
20        createCommitment(new Commitment(initiator, groupParticipant, "accept OR reject"));
21      }
22    }
23  }
```

Listing 2. Detach on deadline expiration.

```
1  public class CNPArtifact extends ProtocolArtifact {
2    private long max_time_millis = 10000;
3    // ...
4    @OPERATION
5    public void cfp(Task task, RoleId initiator) {
6      // same logic
7      execInternalOp("startTiming");
8    }
9    @INTERNAL_OPERATION
10   public void startTiming() {
11     await_time(maxTime);
12     acceptingProposals = false;
13     RoleId groupParticipant = new RoleId(PARTICIPANT_ROLE, RoleId.GROUP_ROLE);
14     createCommitment(new Commitment(initiator, groupParticipant, "accept OR reject"));
15   }
16 }
```

Notice that the above is always the code of the artifact and that there is no need to modify the agent code because the social event did not change.

5. CONCLUSIONS AND DISCUSSION

STS are a challenging frontier of system realization. They answer to a specific need of the contemporary society [Sommerville 2010]: the need of developing software systems that are capable of supporting independent and autonomous stakeholders, each with his/her own objectives, in their interaction. More and more often, such systems build on top (and supervise the use of) resources as diverse as application systems, mobile devices, sensors, or even domotic systems. Stakeholders are proactive and characterized by an own decisional capability but at the same time they have an interest in that the system keeps on working.

We agree with [Singh 2014] that in order to implement STS, it is necessary to realize some kind of *self-governance*, i.e. to realize normative systems, where the actors accept a set of regulations which will guide their behaviour. Although MAS supply the means for supporting the development of distributed systems involving many actors, currently none of the implemented infrastructures allows the realization of self-governance. In this work, we have proposed 2COMM4JADE, an infrastructure for achieving this very objective. 2COMM4JADE is based on JADE, for what concerns the support to the realization of interacting agents, and integrates self-governance mechanisms by relying on the reification of commitments and of commitment-based protocols. These are, at all respects, resources that are made available to stakeholders and that are realized by means of artifacts. The article also shows how to pass from a cross-organizational business process to a set of reified protocols, which represent it

and that will enable the interaction. Recently, we developed on top of 2COMM4JADE a commitment-based typing system [Baldoni et al. 2014]. Such typing includes a notion of compatibility, based on subtyping, which allows for the safe substitution of agents to roles along an interaction that is ruled by a commitment-based protocol. Type checking can be done dynamically when an agent enacts a role.

The proposal is characterized, on the one hand, by the flexibility and the openness that are typical of MAS, and, on the other, by the modularity and the compositionality that are typical requirements of the methodologies for design and development. One of the strong points of the proposal 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. This is a difference with respect to JADE where no decoupling occurs: a pattern of interaction is projected into a set of JADE behaviours, one for each role. Binding the interaction to ad-hoc behaviours does not allow having a global view of the protocol and complicates its maintenance.

Decoupling is an effect of explicitly representing the topmost level of STS and, in particular, of considering social relationships as resources: agent behaviour is, thus, defined based on the existing social relationships and not on the process by which they are created. 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 difference with tuple spaces and CArtAgO itself is that 2COMM4JADE artifacts reify social relationships as commitments, giving them that normative value that, jointly with the fact that by using an artifact an agent explicitly accepts the regulations encoded by it, enables the generation of expectations about the agents behaviour (as advised in [Chopra and Singh 2009]). As such, it allows agents to reason and take decision also based on the others' expected behaviour. For instance, the presence of a commitment *C(i, p, propose, accept* $\vee$ *reject)* grants the participant that its proposal will receive a feedback (it will either be accepted or rejected); if not, the initiator will be liable of a violation. In a tuple space, instead, agents can just read or write the shared blackboard but cannot have expectations on the behaviour of their parties.

A common way to model user activities is by means of *business processes*, that the performers have to follow for achieving some result. Business processes, in their classic definition, are sets of structured, ordered tasks which impose a strict arrangement onto subtask execution: diverging from it results in a failure. The *de facto* standard notation for business processes is provided by the Business Process Model and Notation (BPMN) [White 2004], a graphical language that models a business process as a decorated, enriched flowchart. Although BPMN is a preferred choice for modeling single-actor activities, and even if its latest release (BPMN 2.0) introduces support for a *message* element, it still suffers from major drawbacks in modeling cross-organizational business processes [Desai et al. 2009] and, in general, systems with different stakeholders. The reason relies in the procedural/imperative nature of BPMN: a step-by-step process representation does not consider the autonomy of the interacting entities and hinders their decisional capabilities, when it is used to define both interaction *and* how it must be carried out.

This work has also relationships with works concerning e-institutions, like [Fornara et al. 2007; Fornara et al. 2008; Okouya et al. 2013]. E-institutions introduce a normative aspect that lies on a different level with respect to the one we capture with commitments. Indeed, in our proposal the normative layer stands in the observational semantics of the social actions. We mean to extend the regulative aspects, as a first step, by enriching commitment protocols as proposed in [Marengo et al. 2011; Baldoni et al. 2013]. More in general, we mean to move towards a representation of business

processes as suggested in the Comma methodology [Telang and Singh 2012], where commitment patterns regulate activities instead of being used for giving the action semantics. The advantage of our proposal is that it allows the direct use of the well-known JADE Methodology in the development of STS [Baldoni et al. 2013]. Another advantage is that artifacts are an actual system for implementing mediated interaction, they supply efficient, simple, and direct means for checking the powers for performing institutional actions, and they offer runtime mechanisms for letting the agents perceive the state of the interaction.

Concerning organizations, ORA4MAS [Hübner et al. 2010] and JaCaMo are examples of integrations with artifacts. The latter also accounts for BDI agents. The organizational infrastructure of ORA4MAS is based on $Moise^+$. It allows the enforcement and the regimentation of regulations, by defining both a set of conditions to achieve and the roles that are allowed to or that must achieve them. The limit of this approach is that it cannot manage norms that cannot be restricted to goal achievement. Recently, the use of a communication infrastructure based on artifacts has been proposed to define, in an explicit and clear way, interaction in JaCaMo [Rodrigues et al. 2013]. Nevertheless, the proposal does not supply a normative account of communication.

Finally, concerning the realization of cross-organizational business processes and, in particular, of human-oriented workflows, whose nature is intrinsically social and where the notion of commitment plays a fundamental role [Medina-Mora et al. 1993], it is relevant to cite LoST [Singh 2011]: a commitment-based model for defining declarative protocols. It exploits vectors, each associated to one of the interacting agents, containing the local history of the sent/received messages. LoST enables the representation and monitoring of protocols when it is necessary to transfer local knowledge about occurring interactions between the agents. It works as an adapter for message transfer between agents. 2COMM4JADE, instead, provides agents an environment by which they communicate and, if this is requested, they can perform actions which do not amount to utterances but still entail social effects.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for the many interesting comments which helped improving this paper.

REFERENCES

Matteo Baldoni, Cristina Baroglio, and Federico Capuzzimati. 2013. 2COMM: A Commitment-Based MAS Architecture. In *Engineering Multi-Agent Systems - 1st Int. Workshop, EMAS 2013, Revised Selected Papers (Lecture Notes in Computer Science)*, M. Cossentino, A. El Fallah-Seghrouchni, and M. Winikoff (Eds.), Vol. 8245. Springer, St. Paul, MN, USA, 38–57. DOI:http://dx.doi.org/10.1007/978-3-642-45343-4_3

Matteo Baldoni, Cristina Baroglio, and Federico Capuzzimati. 2014. Typing Multi-Agent Systems via Commitments. In *Proc. of the 2nd International Workshop on Engineering Multi-Agent Systems, EMAS 2014, held in conjuction with AAMAS 2014*, M. B. van Riemsdijk, F. Dalpiaz, and J. Dix (Eds.). IFAAMAS, Paris, France, 341–359.

Matteo Baldoni, Cristina Baroglio, Elisa Marengo, and Viviana Patti. 2013. Constitutive and Regulative Specifications of Commitment Protocols: a Decoupled Approach. *ACM Transactions on Intelligent Systems and Technology* 4, 2 (March 2013), 22:1–22:25. DOI:http://dx.doi.org/10.1145/2438653.2438657

Matteo Baldoni, Cristina Baroglio, Elisa Marengo, Viviana Patti, and Alessandro Ricci. 2011. Back to the future: an Interaction-oriented Framework for Social Computing. In *First International Workshop on Requirements Engineering for Social Computing (RESC 2011), held in conjunction with the 19th IEEE International Requirements Engineering Conference*, A. K. Chopra, F. Dalpiaz, and S. O. Lim (Eds.). IEEE Xplore, Trento, Italy, 2–5. DOI:http://dx.doi.org/10.1109/RESC.2011.6046711

Matteo Baldoni, Guido Boella, and Leendert van der Torre. 2007. Interaction between Objects in `powerJava`. *Journal of Object Technology, Special Issue OOPS Track at SAC 2006* 6, 2 (2007), 5–30. http://www.jot.fm/issues/issue_2007_02

Matteo Baldoni, Andrea Omicini, Cristina Baroglio, Viviana Mascardi, and Paolo Torroni. 2010. Agents, Multi-Agent Systems and Declarative Programming: What, When, Where, Why, Who, How? In *Twenty-five Years of Logic Programming in Italy*, A. Dovier and E. Pontelli (Eds.). Lecture Notes in Computer Science, Vol. 6125. Springer, Berlin Heidelberg, 204–230. DOI:http://dx.doi.org/10.1007/978-3-642-14309-0_10

Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. 2005. JADE - A Java Agent Development Framework. In *Multi-Agent Programming: Languages, Platforms and Applications*, R. H. Bordini, M. Dastani, J. JDix, and A. El Fallah-Seghrouchni (Eds.). Multiagent Systems, Artificial Societies, and Simulated Organizations, Vol. 15. Springer, Berlin Heidelberg, 125–147.

Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. 2007. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, West Sussex, England.

Guido Boella and Leendert W. N. van der Torre. 2007. The ontological properties of social roles in multi-agent systems: definitional dependence, powers and roles playing roles. *Artificial Intelligence and Law* 15, 3 (2007), 201–221. DOI:http://dx.doi.org/10.1007/s10506-007-9030-8

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. 2006. A Survey of Programming Languages and Platforms for Multi-Agent Systems. *Informatica (Slovenia)* 30, 1 (2006), 33–44.

Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, West Sussex, England.

Frances M. T. Brazier, Barbara Dunin-Keplicz, Nicholas R. Jennings, and Jan Treur. 1997. DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. *Int. J. Cooperative Inf. Syst.* 6, 1 (1997), 67–94. DOI:http://dx.doi.org/10.1142/S0218843097000069

Paolo Bresciani and Paolo Donzelli. 2004. A Practical Agent-Based Approach to Requirements Engineering for Socio-technical Systems. In *Agent-Oriented Information Systems, 5th Int. Bi-Conference Workshop, AOIS 2003, Revised Selected Papers (Lecture Notes in Computer Science)*, P. Giorgini, B. Henderson-Sellers, and M. Winikoff (Eds.), Vol. 3030. Springer, Riga, Latvia and New York, NY, USA, 158–173. DOI:http://dx.doi.org/10.1007/978-3-540-25943-5_11

Albert Cherns. 1976. Principles of Socio-Technical Design. *Human Relations* 2 (1976), 783–792.

Amit K. Chopra. 2009. *Commitment Alignment: Semantics, Patterns, and Decision Procedures for Distributed Computing*. Ph.D. Dissertation. North Carolina State University, Raleigh, NC.

Amit K. Chopra and Munindar P. Singh. 2009. An Architecture for Multiagent Systems: An Approach Based on Commitments. In *Proc. of the AAMAS Workshop on Programming Multiagent Systems (ProMAS)*. IFAAMAS, Budapest, Hungary, 184–202.

Rosaria Conte, Cristiano Castelfranchi, and Frank Dignum. 1999. Autonomous Norm Acceptance. In *Intelligent Agents V, Agent Theories, Architectures, and Languages, 5th Int. Workshop, ATAL '98 (Lecture Notes in Computer Science)*, J. P. Müller, M. P. Singh, and A. S. Rao (Eds.), Vol. 1555. Springer, Paris, France, 99–112.

Fabiano Dalpiaz, Amit K. Chopra, and Soo Ling Lim. 2011. *Proc. of the 1st Int. Workshop on Requirements Engineering for Social Computing*. IEEE, Trento, Italy. DOI:http://dx.doi.org/10.1109/RESC.2011.6046710

Mehdi Dastani, Davide Grossi, John-Jules Ch. Meyer, and Nick A. M. Tinnemeier. 2009. Normative Multi-agent Programs and Their Logics. In *Knowledge Representation for Agents and Multi-Agent Systems, 1st Int. Workshop, KRAMAS 2008, Revised Selected Papers (Lecture Notes in Computer Science)*, J.-J. Ch. Meyer and J. Broersen (Eds.), Vol. 5605. Springer, Sydney, Australia, 16–31. DOI:http://dx.doi.org/10.1007/978-3-642-05301-6_2

Nirmit Desai, Amit K. Chopra, and Munindar P. Singh. 2009. Amoeba: A Methodology for Modeling and Evolving Cross-organizational Business Processes. *ACM Trans. Softw. Eng. Methodol.* 19, 2, Article 6 (Oct. 2009), 45 pages. DOI:http://dx.doi.org/10.1145/1571629.1571632

European Parliament. 2004. Directive 2004/39/EC of the European Parliament and of the Council of 21 April 2004 on markets in financial instruments. *Official Journal of the European Union* L145 (2004), 1–44.

Michael Fisher, Rafael H. Bordini, Benjamin Hirsch, and Paolo Torroni. 2007. Computational Logics and Agents: A Road Map of Current Technologies and Future Trends. *Computational Intelligence* 23, 1 (2007), 61–91.

Nicoletta Fornara, Francesco Viganò, and Marco Colombetti. 2007. Agent communication and artificial institutions. *Autonomous Agents and Multi-Agent Systems* 14, 2 (2007), 121–142. DOI:http://dx.doi.org/10.1007/s10458-006-0017-8

Nicoletta Fornara, Francesco Viganò, Mario Verdicchio, and Marco Colombetti. 2008. Artificial institutions: a model of institutional reality for open multiagent systems. *Artificial Intelligence and Law* 16, 1 (2008), 89–105. DOI:http://dx.doi.org/10.1007/s10506-007-9055-z

Foundation for Intelligent Physical Agents. 2002. FIPA Specifications. (2002). http://www.fipa.org.

Jomi Fred Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. 2010. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems* 20, 3 (2010), 369–400.

Elisa Marengo, Matteo Baldoni, Cristina Baroglio, Amit K. Chopra, Viviana Patti, and Munindar P. Singh. 2011. Commitments with Regulations: Reasoning about Safety and Control in REGULA. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011*, K. Tumer, P. Yolum, L. Sonenberg, and P. Stone (Eds.), Vol. 2. IFAAMAS, Taipei, Taiwan, 467–474.

Viviana Mascardi, Maurizio Martelli, and Leon Sterling. 2004. Logic-Based Specification Languages for Intelligent Software Agents. *Theory and Practice of Logic Programming* 4, 4 (2004), 429–494. DOI:http://dx.doi.org/10.1017/S1471068404002029

Raul Medina-Mora, Terry Winograd, Rodrigo Flores, and Fernando Flores. 1993. The Action Workflow Approach to Workflow Management Technology. *Inf. Soc.* 9, 4 (1993), 391–404. DOI:http://dx.doi.org/10.1080/01972243.1993.9960152

Daniel Okouya, Nicoletta Fornara, and Marco Colombetti. 2013. An Infrastructure for the Design and Development of Open Interaction Systems. In *Engineering Multi-Agent Systems - 1st Int. Workshop, EMAS 2013, Revised Selected Papers (Lecture Notes in Computer Science)*, M. Cossentino, A. El Fallah-Seghrouchni, and M. Winikoff (Eds.), Vol. 8245. Springer, St. Paul, Minnesota, USA, 215–234. DOI:http://dx.doi.org/10.1007/978-3-642-45343-4_12

Andrea Omicini, Alessandro Ricci, and Mirko Viroli. 2008. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 17, 3 (2008), 432–456.

Andrea Omicini and Franco Zambonelli. 1998. TuCSoN: a Coordination model for Mobile Information Agents. IDI-TR-5/98. In *Proc. of the 1st Int. Workshop on Innovative Internet Information Systems (IIIS'98)*, D. G. Schwartz, M. Divitini, and T. Brasethvik (Eds.). IDI – NTNU, Trondheim (Norway), Pisa, Italy, 177–187.

Maja Pesic and Wil M. P. van der Aalst. 2006. A Declarative Approach for Flexible Business Processes Management. In *Business Process Management Workshops, BPM 2006 Int. Workshops (Lecture Notes in Computer Science)*, J. Eder and S. Dustdar (Eds.), Vol. 4103. Springer, Vienna, Austria, 169–180. DOI:http://dx.doi.org/10.1007/11837862_18

Daniele Porello, Francesco Setti, Roberta Ferrario, and Marco Cristani. 2013. Multiagent Socio-Technical Systems: An Ontological Approach. In *Coordination, Organizations, Institutions, and Norms in Agent Systems IX, Revised Selected Papers (Lecture Notes in Computer Science)*, T. Balke, F. Dignum, M. B. van Riemsdijk, and A. K. Chopra (Eds.), Vol. 8386. Springer, St. Paul, MN, USA and Dunedin, New Zealand, 42–62. DOI:http://dx.doi.org/10.1007/978-3-319-07314-9_3

Alessandro Ricci, Michele Piunti, and Mirko Viroli. 2011. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems* 23, 2 (2011), 158–192. DOI:http://dx.doi.org/10.1007/s10458-010-9140-7

Thiago Fredes Rodrigues, Antônio Carlos da Rocha Costa, and Graçaliz Pereira Dimuro. 2013. A Communication Infrastructure Based on Artifacts for the JaCaMo Platform. In *Proc. of the 1st Int. Workshop on Engineering Multi-Agent Systems, EMAS 2013, held in conjuction with AAMAS 2013*, M. Cossentino, A. El Fallah Seghrouchni, and M. Winikoff (Eds.). IFAAMAS, St. Paul, Minnesota, USA, 97–111.

Munindar P. Singh. 1999. An Ontology for Commitments in Multiagent Systems. *Artificial Intelligence and Law* 7, 1 (1999), 97–113.

Munindar P. Singh. 2000. A Social Semantics for Agent Communication Languages. In *Issues in Agent Communication (Lecture Notes in Computer Science)*, F. Dignum and M. Greaves (Eds.), Vol. 1916. Springer, Berlin Heidelberg, 31–45. DOI:http://dx.doi.org/10.1007/10722777_3

Munindar P. Singh. 2011. LoST: Local State Transfer - An Architectural Style for the Distributed Enactment of Business Protocols. In *IEEE International Conference on Web Services, ICWS 2011*. IEEE Computer Society, Washington DC, USA, 57–64. DOI:http://dx.doi.org/10.1109/ICWS.2011.48

Munindar P. Singh. 2014. Norms As a Basis for Governing Sociotechnical Systems. *ACM Transactions on Intelligent Systems and Technology* 5, 1, Article 21 (Jan. 2014), 23 pages. DOI:http://dx.doi.org/10.1145/2542182.2542203

Ian Sommerville. 2010. *Software Engineering* (9 ed.). Addison-Wesley, Harlow, England.

Ian Sommerville, Dave Cliff, Radu Calinescu, Justin Keen, Tim Kelly, Marta Z. Kwiatkowska, John A. McDermid, and Richard F. Paige. 2012. Large-scale complex IT systems. *Commun. ACM* 55, 7 (July 2012), 71–77. DOI:http://dx.doi.org/10.1145/2209249.2209268

Pankaj R. Telang and Munindar P. Singh. 2010. Abstracting and Applying Business Modeling Patterns from RosettaNet. In *Service-Oriented Computing - 8th International Conference, ICSOC 2010 (Lecture Notes*

in Computer Science), P. P. Maglio, M. Weske, J. Yang, and M. Fantinato (Eds.), Vol. 6470. Springer, San Francisco, CA, USA, 426–440. DOI:http://dx.doi.org/10.1007/978-3-642-17358-5_29

Pankaj R. Telang and Munindar P. Singh. 2011. Specifying and Verifying Cross-Organizational Business Models: An Agent-Oriented Approach. *IEEE Transactions on Services Computing* 5, 3 (2011), 305–318.

Pankaj R. Telang and Munindar P. Singh. 2012. Comma: a commitment-based business modeling methodology and its empirical evaluation. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012,* W. van der Hoek, L. Padgham, V. Conitzer, and M. Winikoff (Eds.). IFAAMAS, Valencia, Spain, 1073–1080.

Alexander Thiele, Thomas Konnerth, Silvan Kaiser, Jan Keiser, and Benjamin Hirsch. 2009. Applying JIAC V to Real World Problems: The MAMS Case. In *Multiagent System Technologies, 7th German Conference, MATES 2009 (Lecture Notes in Computer Science)*, L. Braubach, W. van der Hoek, P. Petta, and A. Pokahr (Eds.), Vol. 5774. Springer, Hamburg, Germany, 268–277. DOI:http://dx.doi.org/10.1007/978-3-642-04143-3_29

Eric Trist. 1981. The Evolution of Socio-technical Systems: A Conceptual Framework and an Action Research Program. *Occasional paper* 2 (1981).

Danny Weyns, Andrea Omicini, and James Odell. 2007. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems* 14, 1 (2007), 5–30. DOI:http://dx.doi.org/10.1007/s10458-006-0012-0

Stephen A. White. 2004. *Introduction to BPMN*. Technical Report. Object Management Group.

Brian Whitworth and Adnan Ahmad. 2013. Socio-Technical System Design. In *The Encyclopedia of Human-Computer Interaction, 2nd Ed.*, Mads Soegaard with Rikke Friis Dam (Ed.). The Interaction Design Foundation, Aarhus, Denmark, Chapter 24. http://www.interaction-design.org/encyclopedia/socio-technical_system_design.html

Pinar Yolum and Munindar P. Singh. 2002. Commitment Machines. In *Intelligent Agents VIII, 8th International Workshop, ATAL 2001, Revised Papers (Lecture Notes in Computer Science)*, J.-J. Ch. Meyer and M. Tambe (Eds.), Vol. 2333. Springer, Seattle, WA, USA, 235–247. DOI:http://dx.doi.org/10.1007/3-540-45448-9_17

Received ??; revised ??; accepted ??