The Interaction as an Integration Component for the JaCaMo Platform

Maicon R. Zatelli[1], Jomi F. Hübner[1]

Department of Automation and Systems Engineering
Federal University of Santa Catarina (UFSC) – Florianópolis, SC – Brazil
xsplyter@gmail.com, jomi.hubner@ufsc.br

Abstract. Interaction is a subject widely investigated in multi-agent systems (MASs), but some issues are still open. While most of current approaches of interaction in MAS just consider the interaction between agents, some problems are better modeled when the MAS is composed of agents, environment, interaction, and organization. In our approach, we integrate the interaction with the other MAS components, like the organization and the environment, keeping it as a first class abstraction. In this paper we present a conceptual model for the interaction component, a programming language to specify the interaction, and how our approach was integrated in an MAS platform. The main result of this paper is the conception of the interaction as a first class abstraction considering an MAS composed of agents, environment, interaction, and organization.

1 Introduction

It is quite common in MAS that the agents need to interact to achieve their goals. Sometimes an MAS can be composed of *Agent*, *Environment*, *Interaction*, and *Organization* as introduced in [15,23]. In this kind of MAS, the interaction does not concern only the agents, it is strongly related to the environment and the organization of the system. For instance, besides interacting directly with other agents, agents also interact with objects in their environment by means of acting and sensing.

Many works already exist about agents, organization, and environment. There are tools to specify, develop, and execute each of these components. For example, an MAS developer is able to build the environment by means of CArtAgO [40], the organization by means of AGR [21], ISLANDER [20], Moise [27], and so forth, and finally, the agents by means of GOAL [24], JADE [11], 2APL [13], Jason [10], and so on. There are also tools to link these components to work together, such as EIS [5] and JaCaMo [9]. This separation of concerns can improve the maintenance, modularity, organization, reuse of code, etc. It is also easy to see that each of these components can be programmed by different developers, which also facilitates the division of tasks.

In addition, several approaches defend the idea of keeping the interaction as a first class abstraction [14, 32–34, 43, 44]. However, none of current works (Sec. 2) provide us features to specify and execute the interaction considering the existence of the other MAS components, that is, to allow the specification, development, and execution of the interaction not considering only agents, but also considering the environment and the organization.

We already introduced a conceptual model and a programming language for the interaction considering the other MAS components in previous works [49, 50]. In this paper we focus on the integration of the interaction with the JaCaMo platform. JaCaMo is a project that allows the developer to consider each one of the MAS components as first class abstraction. Although the agent, environment, and organization components are already considered by this platform, the interaction component was not properly integrated. In this platform, the interaction is not a first class abstraction, it is simply reduced to messages coded inside the agents program. For instance, it is not easy to find in the code how the system interaction is programmed (it is indeed spread in several agent programs).

The aim of our work about interaction (conceptual model (Sec. 3), programming language (Sec. 4), and integration with JaCaMo (Sec. 5)) is to provide a mechanism to institutionalize how the agents may interact with the different elements in an MAS to achieve the organizational goals. We are linking the organization (e.g. its goals) to the agents (that should fulfill them) and to the environment (by defining interaction protocols that could be used as guidelines for the achievement of the goals). By considering the interaction with the environment, we can formalize more general situations in a protocol, where the agents should interact with the environment by means of performing actions and perceiving changes. We are looking for an interaction component that is able to deal with the other three MAS components. It means that we are considering a more complex MAS, composed of *Agent*, *Environment*, *Interaction*, and *Organization*. The main results that we got with our approach are detailed in Sec. 6 while further works and conclusions are presented in Sec. 7.

2 Related Work

In this section, we present the interaction problematic and some related work. We start with the works focused on interaction between agents, followed by those that consider the interaction with the environment, and in the following, the works that regard the interaction with the organization. Finishing this section, we mention some works that have already introduced the interaction problematic considering the integration with the three other components.

2.1 Interaction and Agents

There are several drawbacks of specifying the interaction inside of the agents code [19, 32, 44]. One of them is related to the maintainability of the system. If the interaction specification is modified, it is necessary to update the code of each agent involved. Another one is related to the protocol composition. The protocols could not be composed at run-time in order to allow more complex interactions.

As pointed by some approaches, it is unnecessary to keep the interaction control inside the agents code [31–33, 35, 44]. The separation of the two issues simplifies the development of applications, leading to a modular approach [22]. Consequently, protocols can be used to compose more complex protocols [12, 16, 17, 28, 37, 38]. In [28, 38, 44], other advantages of a modular approach are presented such as the specification of reusable protocols, the improvements in the validation process, and the capacity to share protocols between agents at run-time.

2.2 Interaction and Environment

One of the main limitations in most of works is to consider the interaction only by means of message exchange between agents, not considering the agent interaction with the environment [2, 3, 6]. Some examples that justify this kind of interaction are presented in [2,3]. One of these examples refers to the election in the human world. When people have to do an election, they do not say the candidate name. They use their hands to interact with the electronic ballot box or simply raise them without saying any word. On the one hand, the electronic ballot box is responsible for computing the votes and notify the winner. On the other hand, by raising their hands, people also may discover the winner of some election only by counting the upper hands. In both cases, the interaction occurs by actions and percepts in the environment and not by speech acts.

There are some works that consider the relation between interaction and environment. In [39] and [42], it is presented a model that allows some different kinds of interaction, called overhearing, or eavesdropping. In this kind of interaction, the agent intercepts messages of others by using the environment. The environment is a way to send and receive messages. In [30], the aim is to conceive an environment as a way to allow indirect interaction. Their focus is on interaction like stigmergy, which is the interaction used by several natural systems such as amoebae and ants. In [35], the environment is considered as a mediator between agents and not a proper first class abstraction. In [31], the environment is also considered by another perspective: the agents could recognize other agents by the concept of neighborhood. The agents are only able to communicate with others depending on how far they are from each other. In both cases [31, 35], it is not considered the actions or percepts performed by the agents in the environment. Moreover, although their approach consider the concept of *roles*, such roles are not related to organizational roles. As a consequence, the specification may lack coherence since different role conceptualization may exist in different components. A role, for example, while existing inside a protocol, may not exist in the organization.

The MERCURIO framework [2,3], a very similar work to ours, focus on integration of the interaction model regarding agents and environment. The environment considers the actions performed by the agents and the percepts that the agents may sense. However, the main aim of MERCURIO is to deploy the interaction with the environment, thus the interaction is also not strongly connected with the organization. As in [31, 35], the roles in the interaction are not the same roles as in the organization.

Finally, in [4, 41] the authors use artifacts to handle the interaction between the agents. In [41], the aim is to provide a communication infrastructure based on artifacts. The implementation of such infrastructure is done in JaCaMo platform [9] and the authors provide the representation of two kinds of artifacts. The former has the aim to represent the interaction protocol itself and allows the specification of a sequence of messages. The latter defines each speech act individually. In [4], the authors use CArtAgO artifacts to embed commitment-protocols following the model introduced in [46–48]. Their work also enriches the JADE [7] with mechanisms to exploit the use of commitments and protocols based on commitments. Each artifact keeps a social state, which is composed of social facts and commitments. Thus, the agents are able to reason about the interaction by means of observing the social state evolution. In both cases [4, 41], instead of the agents exchange messages directly, they use the operations provided by the artifacts. For example, in the contract-net protocol, the operations of the artifact can be `cfp`, `propose`, `refuse`, `accept`, `reject`, `done`, and `failure`. Moreover, the communication artifacts have the aim to notify the receiver about the messages.

2.3 Interaction and Organization

The relation between interaction and organization is also important. The GAIA methodology [45], for instance, has already defined a role as a composition of four main attributes: responsibilities, permissions, activities, and protocols. The protocols are responsible for specifying the interaction between the agents that are playing the organizational roles.

Some works about organization already relate the interaction with the organization by means of a dialogical dimension [8, 18, 20, 21, 25]. In this case, they use several organizational concepts, like goals, roles, and obligations. Each of these concepts is strongly connected with the interaction concepts, which means that, for example, the roles in the interaction are the same roles as in the organization.

2.4 Summary

Although none of the presented works addresses the integration of the interaction with the three other components in order to allow its specification, development, and execution, some works already address this topic in an AOSE perspective. MAS-ML [43] and O-MaSE [14] are a modeling language and methodology, respectively, which consider the interaction integration with the three other components. However, both approaches are conceived for the specification phase and do not consider the implementation and execution phases. In addition, even providing tools to generate code, they do not generate the interaction code.

Thus, even if some authors are concerned with the interaction between agents and some of the other components, none of them integrates the interaction with the three components in a unified perspective and consider the development and execution phases. Moreover, in some of them, the interaction specification is conceived to be handled by humans during the MAS design and does not allow the agents to read it (or eventually to change it) at run-time. By not considering the interaction as a first class entity and by lacking an integration with the other three components, we may have a series of difficulties in the development of some applications: (i) it is difficult to have an overall view of the interaction in the MAS because the interaction code is spread in several parts of the system (e.g. it is only possible to see the interaction if we open the code of each agent involved in the interaction); (ii) the MAS developer is not able to formalize (by means of a programming language) the expectations about the MAS evolution considering the interaction both with the environment and with other agents; (iii) it is not possible to provide a more detailed specification for the agents to help them to achieve organizational goals, especially if the agents need to interact with the environment; and

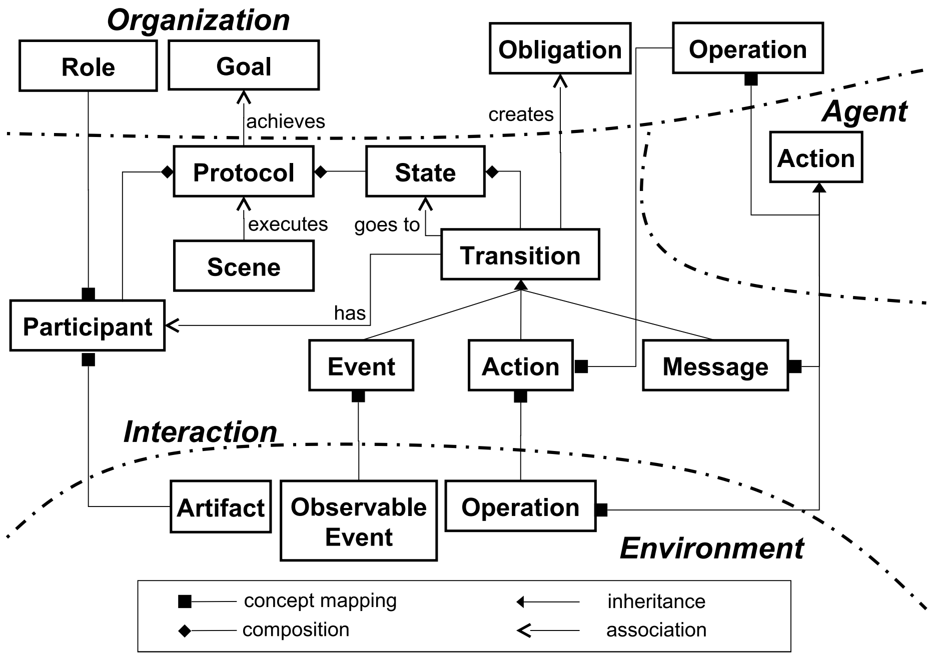

Fig. 1: Conceptual model.

(iv) agents in open systems have more difficulties to interact because the interaction protocols are not explicitly specified and available at run-time.

3 Conceptual Model

This section briefly presents how the several MAS components are conceptually integrated with the interaction. Only the core ideas of the model are described here. More details can be found in [49].

Fig. 1 shows the four MAS components and the relations between the interaction and the others. In order to keep the figure clear and clean, we only show the concepts that were directly related to the interaction. The most important concept in our model is the interaction protocol[1], which is basically composed of a set of participants, transitions, states, and goals. Each transition links two states (one source state and one target state) and it can be fired by an event, a message, or an action. When some transition is fired, a new state is achieved and the protocol execution progresses. In order to separate the protocol specification and the protocol execution, we call *scene* an instance of a protocol. It is possible for a protocol to have several scenes executing at the same time.

The organizational concepts used in our model (top of Fig. 1) are based on the organizational models presented in [18,27]. The interaction is related to organization in

[1] In this paper, we consider the definition of protocol presented in [29].

four points. Firstly, the protocols are related to organizational goals. A protocol specifies a possible interaction scheme to achieve them. When a protocol finishes successfully, the organizational goal is considered achieved. For example, if there is an organizational goal for an agent to contract a company to build a house, such goal can be achieved by the use of a contract-net protocol. The protocol is just *one* (and not the *only* or even a *mandatory*) way for the agents to achieve the organizational goals. It can exists several protocols to achieve the same goal and the agents could also achieve a goal using other means. We could also imagine the existence of protocols without a relation to organizational goals, however, in this work, our main objective with the use of protocols is to help the agents to achieve the organizational goals. Thus, we are not interested in the representation of protocols that do not drive the agent to accomplish organizational goals and neither about what the agents do for achieving their own (not organizational) goals.

The second organizational concept used in our model is obligation. The transitions of a protocol are related to organizational obligations. Obligations are created for the agents to perform the action that fires some enabled transition of the scene and thus evolve its execution. For example, if there is a transition in a protocol that specifies that some agent needs to tell the price of a product to another agent, an obligation with this information will be created as soon as the transition is enabled. Furthermore, the use of obligations does not hinder the agents from trying other means to achieve the goals. The agents are free to violate them.

Thirdly, the participants of a protocol are related to organizational roles. To be a participant in a protocol, an agent must previously play a role in the organization (e.g. the role `baker` or `manager`). Since the organization constraints the role adoption based on the agent skills, the agent will be able to perform the activities required as a participant in the protocol. Finally, the organization also provides operations, which are the actions that some agent can perform in the organization such as adopts or leaves some role, commits to some mission or goal, and achieves some goal.

The environmental concepts used in our model (bottom of Fig. 1) are based on the A&A meta-model introduced in [36]. We map the concept of artifact onto a participant in the interaction component, which constrains the participation of artifacts in the protocol; the operations, which represent the actions that the agents can perform in the environment (for example, the agent can execute actions to regulate the temperature of an oven, such as turns the oven on or off); and finally, the observable events, which agents can perceive in the environment, such as an alarm indicating that the temperature of an oven is too high, the color of something, the sound of a machine, etc. It is important to notice that the artifacts are not an autonomous entity and, in our approach, we are not trying to define what the *artifacts* should do. Rather, the protocol defines which actions the *agents* should do on them. Besides the actions, the use of protocols is a way to handle the observable events that are being produced by the artifacts.

The agent component (right side of Fig. 1) provides the concepts of action, which can be some action performed in the environment or in the organization, and the message exchange, which represents the use of communicative acts (e.g. `tell`, `achieve`) in order to interact with the other agents. The actions that the agents perform in the environment or in the organization are mapped onto their respective concepts in their respective components. An action performed by the agent in the organization is mapped onto the concept of action in the organization component while an action performed by the agent in the environment is mapped onto the concept of action in the environment component. Finally, the concept of message exchange is directly mapped onto the concept of message in the interaction component.

The conceptual model introduced in this section is a generic solution for the integration of the organization, environment, and agents based on the concepts depicted in Fig. 1. For example, if the organization provides concepts like goals, roles, and obligations, it can fit very well in the proposed model. Moreover, the model can also be adapted to other organizations, environments, or agents. One of the core ideas of this paper is to take advantage of using a formal representation for the interaction considering the environment and the organization. A well-detailed protocol (specified by means of messages, actions, and events) can help the development of open systems or help the agents that do not know how to achieve some organizational goal. Thus, protocols are used to define a more general behavior for a *system* and not simply to define the behavior of the agents using message exchange.

4 A Language to Specify Interaction Protocols

In this section, we map the concepts presented in Fig. 1 onto a programming language used to specify interaction protocols.[2] The language is mostly presented by means of two examples. The aim of the first example is to provide a typical sequence of steps used to write a protocol in our approach. For this first example, we consider a simplified situation where an agent must make a cake. The protocol shows especially how an agent interacts with the environment by means of actions and percepts. The second example illustrates more features of the language, such as the specification of message exchanges and timeouts. In both examples, we present very simple situations, however the real advantages of the proposed interaction protocols are better noticed in large MAS, where the system is composed of hundreds of agents with complex tasks and interactions.

The first step to build a protocol with the proposed language is to decide which organizational goals the protocol must achieve. For example, to make a cake for a bakery organization, we can conceive a protocol as a way to achieve the goal "*to make a cake*". When the cake is done, the goal "*to make a cake*" can be set as achieved.

In the following, we need to decide who will be the participants of the protocol. Using the example of the cake, we can assume that in the bakery organization there are the roles *baker* and *cake_decorator*. The *baker* is responsible for the cake production while the *cake_decorator* is responsible for the cake decoration. Therefore we can define `baker` and *cake_decorator* as participants of the protocol. In addition, we may include some environment elements that will participate of this scenario. For example, we will need an *oven*, a *blender*, a *clock*, etc.

Then we specify the states of the protocol and the order that they should be achieved. The states of a protocol can be achieved by means of transitions that can be fired by

[2] We will only briefly present the most important parts of the language, since more details can be found in [50].

Algorithm 1 Making a cake protocol.

```
protocol making_a_cake {
  description: "Tell the agent how to make a cake";
  goals: "to_make_a_cake";
  participants:
    agBaker agent "baker";
    agCakeDecorator agent "cake_decorator";
    artBlender artifact "artifacts.Blender";
    artOven artifact "artifacts.Oven";
    artClock artifact "artifacts.Clock";
  states:
    n1 initial; n2; n3; n4; n5; n6 final;
  transitions:
    n1 - n2 # agBaker -- action "mixIngredients" -> artBlender;
    n2 - n3 # agBaker -- action "putCake" -> artOven;
    n3 - n4 # agBaker -- action "setTimer" -> artClock;
    n4 - n5 # artClock -- event "alarm" -> agCakeDecorator;
    n5 - n6 # agCakeDecorator -- action "takeCake" -> artOven;
}
```

actions that the agents perform in the environment, events that the agents can perceive, and messages that the agents can exchange. Back to the *making a cake* scenario we can see some transitions. We can define as a first transition that the agent with the role *baker* needs to mix the ingredients using the *blender*. In the second transition, the *baker* needs to put the cake into the *oven* and finally it needs to set the *clock* with the required time. After the time elapses, the *clock* emits a sound, which can notify the *cake_decorator* that the cake is done. Thus, the *cake_decorator* can take the cake out of the *oven*. In Sec. 5, we give more details about how transitions produce obligations.

Finally, we can define a name, some description, the initial state, and the final states. Notice that we can have several final states, however we can only have one initial state. In the *making a cake* scenario, we can set the initial state as when there is “nothing” of the cake. As a final state, we can set the state after the agent takes the cake out of the oven. Therefore, when this final state is achieved, the goal *to make a cake* is achieved in the organization. A possible implementation of this protocol is presented in Code 1.

The advantage of using protocols in the case of the making a cake scenario is the openness. A new agent, which has never made a cake before, can adopt the role *baker* and follow the protocol specification. The protocol is a way to guide the new agent to make the cake. Therefore, we can replace the agents and if they know how to follow protocols, they can make a cake easily. Another aspect of this example is that we only used actions and events, such as *put the cake into the oven*, *take the cake out of the oven*, *set the time in the clock*, and *the sound emitted by the clock*. Both actions and events are related to environmental concepts. Although the transitions in our example represent macro-tasks, we could detail the protocol as much as we need. For example, the transition `n5 - n6` could be detailed using other actions. Instead of simply taking the cake out of the oven, we could specify that the agent should turn the oven off, open the oven door, take the cake out of the oven, and close the oven door.

Code 2 presents another example of protocol, where the aim is to serve a customer in a store and the sellers do an election in order to decide which one will serve the customer. The participation of the agents is defined in lines 5 and 6, which state that they

Algorithm 2 Attending protocol.

```
 1. protocol attending {
 2.   description: "Serve a customer";
 3.   goals: "chooseSeller";
 4.   participants:
 5.     playerCustomer agent "customer";
 6.     playerSeller agent "seller" all;
 7.     artBallotBox artifact "artifacts.BallotBox";
 8.   states: k1 initial; k2; k3; k4 final;
 9.   transitions:
10.     k1 - k2 # playerCustomer -- message[tell] "needSeller" -> playerSeller;
11.     k2 - k3 # playerSeller -- action "vote(X)" -> artBallotBox
                    : ".string(X) & .is_agent(X)";
12.     k2 - k3 # timeout 30000;
13.     k3 - k4 # artBallotBox -- event "winner(Y)" -> playerSeller;
14. }
```

must play the role `customer` (line 5) or the role `seller` (line 6) in the organization. The protocol also includes the participation of a ballot box artifact to help the agents to vote in an anonymous approach (line 7).

The protocol is composed of four states (line 8): `k1`, `k2`, `k3`, and `k4`, where `k1` is the initial state and `k4` is the final state. On the one hand, the available transition from state `k1` is defined in line 10. It defines that the agent who is playing the participant `playerCustomer` must send a message to the agents who are playing the participant `playerSeller` informing them that it needs some seller. On the other hand, the available transitions from state `k2` are those defined in lines 11 and 12. The former can be triggered only by agents participating as `playerSeller` in the protocol by doing the action `vote(X)` on the artifact `artBallotBox` (the ballot box). The latter is defined with a `timeout` statement (line 12). The `timeout` is important in situations where temporal constraints are fundamental, such as the time that an agent must wait for the proposals of the others in an auction.

An important mechanism used in the language is the unification, which is equivalent with the traditional unification mechanism of several agent languages and also Prolog. When an agent performs the action `vote` or the environment produces the event `winner`, it must unify with their respective expressions `vote(X)` and `winner(Y)`, where X and Y are variables. Notice that in transition `k2 - k3` we have specified the test `".string(X) & .is_agent(X)"` which means that the agent performs the action `vote(X)`, the `X` must be both a `String` and an existing agent in the MAS. Moreover, it is important to notice that this test expression is any `String`, which means we can have many ways to evaluate some action. More details about this mechanism is explained afterwards.

Finally, the last transition of the protocol (line 13) defines that the `artBallotBox` counts the votes and emits an observable event named `winner(Y)`, where `Y` is the winner name. With the successful termination of the protocol, the goal `chooseSeller` is achieved in the organization (line 3).

It is also possible to specify different ways to fire transitions. Fig. 2 presents the language grammar with its non-terminal symbols. The non-terminal `duty` defines what

```
protocol        ::= "protocol" <ID> "{" description
                                        goals
                                        participants
                                        states
                                        transitions "}"
description     ::= ("description" ":" <STRING> ";")?
goals           ::= "goals" ":" (goal)+
goal            ::= <STRING> ";"
participants    ::= "participants" ":" (participant)+
participant     ::= participantId partDescription ";"
partDescription ::= ("agent" role | "artifact" type) partCardinality
partCardinality ::= ("all" | ("min" <INTEGER>)? ("max" (<INTEGER> | "+"))?)
participantId   ::= <ID>
type            ::= <STRING>
role            ::= <STRING>
states          ::= "states" ":" (state)+
state           ::= stateId ("initial" | "final")? ";"
stateId         ::= <ID>
transitions     ::= "transitions" ":" (transition)+
transition      ::= stateId "-" stateId "#" (occurrence | timeout | import)
timeout         ::= "timeout" <INTEGER> ";"
import          ::= "import" <STRING> mapping ";"
mapping         ::= "mapping" "{" (mapFromTo)+ "}"
mapFromTo       ::= participantId participantId ";"
occurrence      ::= pCardOccur "--" duty "->" pCardOccur ((trigger)+ | ";")
pCardOccur      ::= participantId ("[" <INTEGER> "]")?
duty            ::= dutyType <STRING>
dutyType        ::= ("event" | "action" | "message" "[" <ID> "]")
trigger         ::= ("trigger" pattern (":" content)? | ":" content) ";"
pattern         ::= <STRING>
content         ::= <STRING>
```

Fig. 2: Language grammar [50].

Algorithm 3 Reply to call-for-proposals in the contract-net protocol.

```
1. no2 - no3 # seller -- message[tell] "replyCFP(CNPId)" -> client
           trigger "refuse(CNPId)" : ".number(CNPId)";
           trigger "propose(CNPId,Offer)" : ".number(CNPId) & .number(Offer)";
```

must happen to fire the transitions and each transition may have several different verifications (represented by the non-terminal `trigger`) to make sure whether some occurrence is valid to fire it. For example, in Code 3, we specified part of the contract-net protocol. In this part, the agents playing the participant `seller` must answer the call-for-proposals (`replyCFP(CNPId)`) sent by the agent playing the participant `client`. The triggers define the two possible answers that the agents could use to fire the transition `no2 - no3`. The former indicates that the `seller` could refuse to make a proposal (`refuse(CNPId)`), while the latter indicates that the `seller` could send a proposal (`propose(CNPId,Offer)`). In the previous protocols, presented in Code 1 and Code 2, we do not have such kind of situation because for each transition there is only one way to fire it. However, as presented in Code 3, we can represent transitions that could be fired using other ways.

Algorithm 4 Protocol composition.

```
y2 - y3 # import "election.ptl"
                 mapping {
                   employee elector;
                 };
```

The non-terminal `trigger` is composed of an expression to evaluate the occurrence pattern (represented by the non-terminal `pattern`) and an expression to evaluate the occurrence content (represented by the non-terminal `content`). For example, in Code 3, the pattern is represented by `"refuse(CNPId)"` and `"propose(CNPId,Offer)"`, while the evaluation of the content is represented by `".number(CNPId)"` and `".number(CNPId) & .number(Offer)"`, respectively. If the occurrence satisfies the pattern, then we can evaluate the content of the variables (if there are variables in the pattern).

If the `pattern` is omitted, the expression defined in the non-terminal `duty` will be considered as the `pattern`. For example, the pattern is omitted in the case of the protocols presented in Code 1 and Code 2. Considering the transition `k2 - k3` presented in Code 2 (line 11), the expression specified in the `duty` (`vote(X)`) is used as the pattern. Next to the symbol `:` (line 11), it is defined the expression to evaluate the content of the action. Suppose the agent tries to execute something like `vote("Ana",22)`. This action is not valid because it does not unify with the pattern `vote(X)`, then the action is discarded. However, suppose that the agent performs the action `vote(22)`. This action follows the pattern because it unifies the pattern (with `X` = 22), however the action is invalid because 22 is not a `String`, as required by the `content`. Finally, suppose the agent tries to execute the action `vote("Ana")`. We have `X = "Ana"` and `"Ana"` is a `String`. If Ana is also an agent, the action is valid to fire the transition.

Other features of the language are the composition of protocols and the cardinality. The composition is made by using the `import` directive. The `import` directive needs the information about the file of the sub-protocol and a mapping between the participants of the protocol and the sub-protocol. The mapping is necessary because, sometimes, the protocols may not have the same participants. An example of composition is presented in Code 4. In this case, the transition `y2 - y3` will be fired after the `election` protocol be accomplished. The mapping in this protocol is made by defining that the participant `employee` will be the participant `elector` in the election protocol. Although the election protocol needs a goal related to it, during the composition its goal will be ignored. Only the goals related to the main protocol will be used at run-time.

The language also provides two different kinds of cardinality: the participant cardinality and the transition cardinality. The former is related to the required number of entities playing some participant in the protocol. The latter is related to the number of entities that are necessary to perform the duty specified in some transition. For example, we can have several attendants in a call-center, however we just need *one* to answer the phone. In an election, we have electors and it is necessary that *all* of them participate. Therefore, with cardinality mechanisms we can define these situations. Such features are presented in more details in [50].

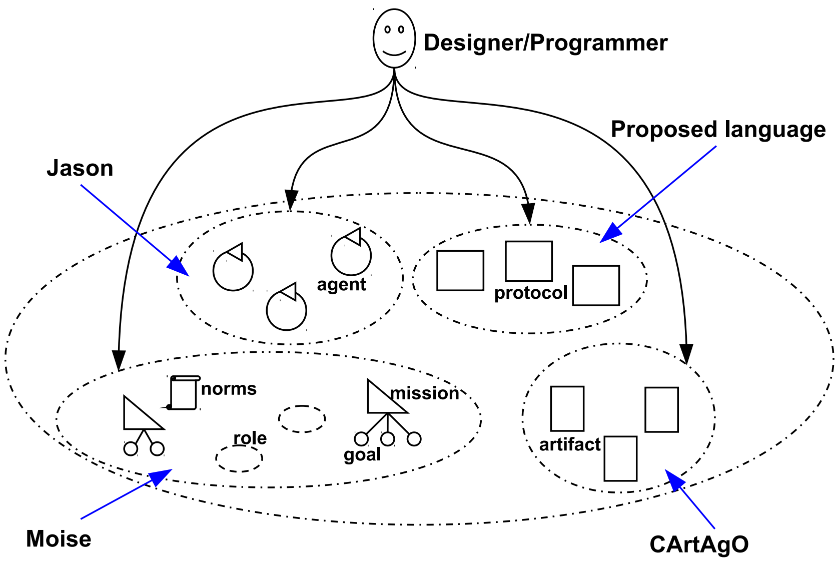

Fig. 3: Concern separation.

5 Integrating with JaCaMo

The main aim of the integration of our interaction approach with JaCaMo is to provide an MAS programming platform supporting concerns separation also considering the interaction.[3] Fig. 3 shows a general idea of the integration. In JaCaMo platform, the MAS developer can already program each of the three components separately and each component can be programmed with specific tools and languages. The organization can be programmed using Moise, the agents can be programmed using Jason, and the environment can be programmed by using CArtAgO. In our work, we also enrich the JaCaMo platform with the interaction component, which also has its proper tool and language. The next two sections detail how the integration was made.

5.1 Mapping the Conceptual Model onto JaCaMo Platform

In order to integrate our approach into JaCaMo platform, we map the model presented in Fig. 1 onto the JaCaMo platform. Since the components of agent, organization, and environment in JaCaMo already use the same concepts, we need to integrate the relations between the interaction component and the other ones. As part of the integration, we introduce an interaction artifact (`SceneArtifact`), which allows the agents to work with the interaction component. A similar integration was already done with the organization by means of ORA4MAS artifacts [26].

[3] The full implementation of our approach can be found at `https://sourceforge.net/projects/intmas/`.

Algorithm 5 Handling the organizational obligations created by the scene artifact.

```
+obligation(MyName, _Scene,
            transition(_CurrentState, _GotoState, _TriggerType, _Target, Duty),
            _Deadline):
  .my_name(MyName)
<-
  !Duty.
```

Basically, when the agent receives an organizational obligation to achieve some organizational goal, it can verify which protocol can be used to help the accomplishment of the goal. The agent can instantiate the protocol by informing its specification. Each instance of a protocol is executed in a different instance of the `SceneArtifact`, which allows the agent to follow the execution of each scene individually. The `SceneArtifact` reads the protocol specification and convert it in several observable properties to guide the agents during the scene execution.

The relation between the protocol and the organizational goal (Fig. 1) is reified by using a link between the artifact `SceneArtifact` and the artifact `SchemeBoard` of the organization. The artifact `SchemeBoard` is the responsible to deal with the organization goals in the organizational component of JaCaMo. Therefore, when the `SceneArtifact` achieves the final state of a protocol, it changes the state of the goals related to the protocol in the organization by means of that link.

An important part of our approach is the use of obligations, represented by the relation between transition and obligation (Fig. 1). Everytime the scene achieves a new state, new obligations are created to help the agents to accomplish the protocol. For example, suppose the protocol presented in Code 2. When the state `k1` is enabled, an obligation related to the transition `k1 - k2` is created. This obligation defines that the agent playing the participant `playerCustomer` should send a message `needSeller`, using the performative `tell`, to the agents playing the participant `playerSeller`. When the messages are sent, the scene moves from state `k1` to `k2` and the obligation is accomplished. As a consequence, new obligations will be created. In this case, it will be created an obligation related to the transition `k2 - k3` for the agents playing the participant `playerSeller` to perform the action `vote(X)` on the artifact `artBallotBox`. In addition, this new obligation will have a timeout of 30000 milliseconds, as defined in line 12. Although created from a fact in the interaction component, the obligations exist in the organizational component of the MAS.

The agents in JaCaMo already knows how to handle organizational obligations because it is a concept already used in Moise. Thus, it is not necessary to build any new specific mechanism for the agents to work with the obligations created by the interaction component. The main advantage of using obligations is that they are created at run-time, which also means that the protocols can be updated at run-time. For example, if the order of the transitions is modified in the protocol specification, the next obligations will be created respecting the new order of the transitions. Therefore, the agents code usually does not need to be modified all the time that the protocol is modified, since the agents simply follow the obligations.

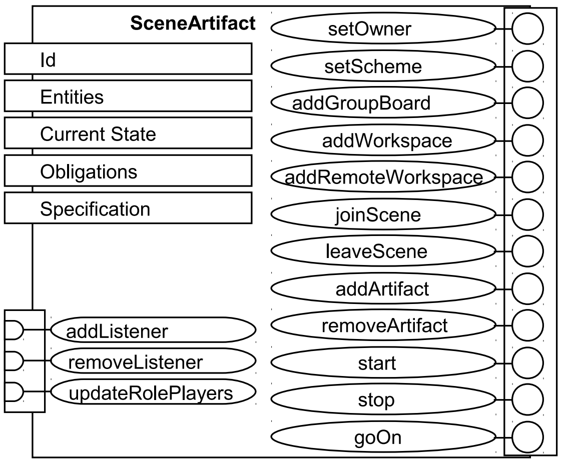

Fig. 4: Scene artifact.

The Jason code presented in Code 5 illustrates how the agents can deal with the obligations created by the interaction component. In line 1, it is indicated that the agents perceive an obligation to do a duty in a certain moment of the scene execution. That duty must be done in order to fire the enabled transition. As soon as the agents perceive that obligation, they create a new goal to accomplish that duty (line 4). Notice that it is only necessary to add the code presented in Code 5 in the agents program to make the agents able to create their own goals to accomplish the duties of the protocol. If the protocol is modified, other obligations for the agents are created and the agents will be able to continue following the protocol in the same way.

Fig. 4 shows the interface of the `SceneArtifact`, with its operations and observable properties. The operations allow the agents to play some participant of the scene (`joinScene`), to leave the scene (`leaveScene`), add and remove artifacts of the scene (`addArtifact` and `removeArtifact`, respectively), and to start (`start`), stop (`stop`), or continue (`goOn`) the scene execution. Moreover, by means of observable properties, the agents can get some information about the scene. For example, they can see the current state of the scene (`Current State`), the enabled transitions (by means of the `Current State` property), their obligations (`Obligations`), the entities that are playing the participants (`Entities`), the protocol specification (`Specification`), etc.

Since CArtAgO uses the concept of *links* to allow the representation of "operations" that can be accessed by other artifacts, we specify some links to allow the development of tools to monitor the scene execution. In that sense, there are links to add and remove some listener (`addListener` and `removeListener`, respectively). The general idea of these links is to allow other artifacts to receive information about the scene evolution. For example, it is possible to get information about the enabled states and transitions, the fired transitions and the actions, messages, and events that were responsible to fire each transition.

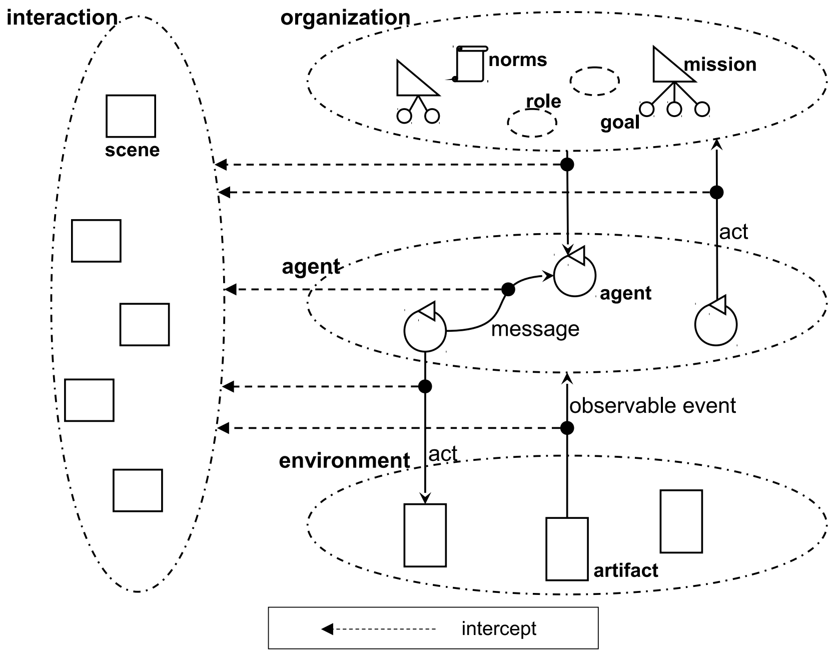

Fig. 5: Interception model.

The last link (updateRolePlayers) is necessary because the interaction mechanism needs to know which are the agents playing each role in the organization. This information is used to handle the cardinalities and to make sure that certain agent is really playing some role. The Moise GroupBoard artifact already provides a link to add listeners and gets such information. In the same way, we need to handle the cardinalities of artifacts and verify if certain artifact is of some kind. Therefore, we created a link (getArtifactList) into the WorkspaceArtifact in CArtAgO. This link has the aim to return the list of all artifacts and their kinds in some workspace. Such mechanisms are introduced to reify the relations between interaction participant with organizational role and environmental artifact, as presented in the conceptual model (Fig. 1).

5.2 Getting Messages, Actions, and Events

All the messages, actions, and events must be intercepted and sent to the scenes. Fig. 5 shows the interception model. It shows messages, actions, and events being intercepted during their occurrences. The agents do not need to notify the scene artifact about what they are doing explicitly, since they could try to cheat the interaction mechanism. For example, they could notify the interaction about things that they have never done.

Some related work use a mediator agent to get the necessary information [1], however the mediator agent is an autonomous entity and then it is possibly malicious. Our approach to get messages, actions, and events is similar to the approach presented in [3, 35], where the authors define a layer that behaves like a filter to consider only the correct messages to change the interaction state. In order to do that in JaCaMo platform, in a first moment, we modified the agent architecture. The new agent architecture intercepts the messages exchanged between the agents, the events that occurs in the environment, and the actions that the agents perform in the environment. Notice that the agents interact with the organization in JaCaMo by means of organizational artifacts in the environment, therefore it is not necessary to create a specific mechanism to deal with the actions performed in the organization. In the end, the messages, actions, and events that were intercepted are delivered to the scenes that the agents are attending. Then, they will be processed and evaluated in order to fire the enabled transitions.

6 Results and Discussion

Our main contribution in this paper is the integration of the interaction component into the JaCaMo platform. With this integration we have an MAS platform to program the agents, the environment, the organization, and the interaction, all of them as first class abstractions. We can now specify the interaction in a separated component, avoiding specifying the interaction inside the code of agents or other components.

As another result, we can also specify the agents more independent of the application. Before the integration of our approach into JaCaMo, it was necessary to specify how the agents interact with the other MAS components in their own code. With the interaction integrated into JaCaMo by means of artifacts and assuming the fact that agents already know how to deal with artifacts and organization, the agents do not need any specific mechanism to deal with the interaction. Even in the case of open and heterogeneous MAS, a global behavior can be defined for the overall system by means of the interaction. It is possible because the interaction allows the definition of the desired sequence of steps to achieve the organizational goals. Moreover, while the organizational goals provide information about *what* the agents need to do, the interaction protocols provide a more detailed description about *how* to behave to achieve them.

The integration with the JaCaMo platform allowed us to evaluate our interaction proposal and also to provide an example of how to integrate it into an MAS platform composed of agents, environment, and organization. In our experiments, we saw several advantages considering the interaction as a first class abstraction. For example, we can update the interaction without changing the code of the other MAS components. We also got some positive results with the relations that we made between the interaction and the other MAS components. For example, the obligations facilitate the agent programming and allow the agents to reason about them, specially whether the agents already can handle organizational obligations, as in the case of JaCaMo platform. We can change the sequence of transitions of protocols and, because the obligations are created in execution time regarding to transitions, we do not need to update the agents code. Moreover, in future works, norms and obligations will allow us to create punishment and reward mechanisms to prevent malicious behavior and reward the agents with good performances. The relation between participant in the interaction and role in the organization allows the agents to search for partners to cooperate because the protocols specify which roles they must interact with. The relation between interaction and environment by means of artifacts permits the specification of how the agents must proceed to interact with the artifacts by means of actions and observable events.

As some drawbacks of the integration with JaCaMo platform, we noticed a decrease in performance and some negative impact related to scalability. In fact, it was an expected impact because we did not focused on performance and scalability issues in this first moment. The main reason for this negative impact is the interception and management of messages, actions, and events that happen in the MAS execution. Since most of them could be relevant to the scenes, after the interception mechanism catch such occurrences we need to send them to the scenes and process them. So far, we built a centralized solution to process such occurrences in each scene, however it seems not the best solution for an MAS where there are many messages exchanges, actions, and events. The improvement of these issues remains as future work.

Another questionable point of our approach is related to the number of different languages that the developer should learn in order to implement an MAS using JaCaMo platform. With the integration of the interaction component into JaCaMo platform, the MAS developer will have four different languages to learn, each one dedicated to specify one of its components (agents, organization, environment, and interaction). Indeed, learning four languages would require more time and investments from the MAS developers. However, all the four languages are more suitable to implement their own concerns. For example, in order to specify the environment, it is better to use a specific environmental language than to specify the environment by means of an agent language. Naturally, when it is necessary to implement a simple MAS, most of times, the agents themselves are enough to solve the problems. The organization, environment, and interaction are better suitable to implement large and complex systems, where the separation of concerns is underlying.

Finally, our approach is not the only one to deal with interaction and some of the other components. As we presented in Sec. 2, there are several approaches of interaction, however, none of them integrate the interaction with all the other three MAS components in a unified way. Some of them handle the interaction between agents, others deal with the interaction and the environment or organization. Furthermore, our proposal is focused on more complex MAS, composed of agents, environment, and organization. Our aim is to integrate these components by means of the interaction and explore the advantages of this kind of MAS.

7 Conclusions and Future Works

In this paper we presented the integration of an approach of interaction considering agents, environment, and organization into the JaCaMo platform. Although we present the integration with the JaCaMo platform, our approach can also be integrated with other MAS platforms. We also highlighted the interaction model and the programming language. As future works, we intend to evaluate the use of this proposal in the development of large systems and also to verify protocols that are created by some agent, since the agents could create protocols at run-time and execute it. Other interesting subjects to explore are how the agents could reason about a protocol in order to optimize its execution, and a proposal of a mechanism to specify and handle exceptions. Finally, mechanisms of punishment and reward should be studied for the purpose of evaluating the performance of the agents when they are participating of some scene.

Acknowledgments. The authors are grateful for the support given by CNPq, grants 140261/2013-3 and 306301/2012-1. We would also like to thank the reviewers for the useful comments and questions, which helped us to improve this paper.

References

1. D. Ancona, S. Drossopoulou, and V. Mascardi. Automatic generation of self-monitoring MASs from multiparty global session types in jason. In *Proc. of DALT*, 2012.
2. M. Baldoni, C. Baroglio, F. Bergenti, A. Boccalatte, E. Marengo, M. Martelli, V. Mascardi, L. Padovani, V. Patti, A. Ricci, G. Rossi, and A. Santi. MERCURIO: An interaction-oriented framework for designing, verifying and programming multi-agent systems. In *Proc. of MALLOW*, pages 134–149, 2010.
3. M. Baldoni, C. Baroglio, F. Bergenti, E. Marengo, V. Mascardi, V. Patti, A. Ricci, and A. Santi. An interaction-oriented agent framework for open environments. In *Proc. of AI*IA*, pages 68–79, Berlin, Heidelberg, 2011. Springer.
4. M. Baldoni, C. Baroglio, and F. Capuzzimati. 2COMM: a commitment-based mas architecture. In *Proc. of the 1st EMAS@AAMAS*, pages 17–32, 2013.
5. T. M. Behrens, K. V. Hindriks, and J. Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, 2011.
6. G. Bel-Enguix and M. D. Jimenez-Lopez. Agent-environment interaction in a multi-agent system: a formal model. In *Proc. of GECCO*, pages 2607–2612, New York, NY, USA, 2007. ACM.
7. F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi. JADE - a java agent development framework. In R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 125–147. Springer, 2005.
8. O. Boissier, F. Balbo, and F. Badeig. Controlling multi-party interaction within normative multi-agent organizations. In *Proc. of MALLOW*, pages 17–32, 2010.
9. O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 2011.
10. R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley, Liverpool, 2007.
11. L. Braubach, E. Pokahr, and W. Lamersdorf. Jadex: A BDI agent system combining middleware and reasoning. In *Ch. of Software Agent-Based Applications, Platforms and Development Kits*, pages 143–168. Birkhaeuser, 2005.
12. L. Cabac, D. Moldt, and H. Rolke. A proposal for structuring Petri net-based agent interaction protocols. In *Proc. of ICATPN*, pages 102–120, Berlin, Heidelberg, 2003. Springer.
13. M. Dastani and J. C. Meyer. A practical agent programming language. In *Proc. of ProMAS*, pages 107–123, Berlin, Heidelberg, 2008. Springer.
14. S. A. DeLoach and J. L. Valenzuela. An agent-environment interaction model. In *Proc. of AOSE*, pages 1–18, Berlin, Heidelberg, 2006. Springer.
15. Y. Demazeau. From interactions to collective behaviour in agent-based systems. In *Proc. of EuroCogSci, Saint-Malo*, pages 117–132, 1995.
16. N. Desai, A. U. Mallya, A. K. Chopra, and M. P. Singh. OWL-P: A methodology for business process development. In *Proc. of AOIS*, pages 79–94, 2005.

17. N. Desai and M. P. Singh. A modular action description language for protocol composition. In *Proc. of AAAI*, pages 962–967. AAAI Press, 2007.
18. V. Dignum, J. Vázquez-salceda, and F. Dignum. Omni: Introducing social structure, norms and ontologies into agent organizations. In *Proc. of PROMAS*, pages 181–198. Springer, 2004.
19. T. Doi, Y. Tahara, and S. Honiden. IOM/T: an interaction description language for multi-agent systems. In *Proc. of AAMAS*, pages 778–785, New York, NY, USA, 2005. ACM.
20. M. Esteva, B. Rosell, J. A. Rodriguez-Aguilar, and J. L. Arcos. AMELI: An agent-based middleware for electronic institutions. In *Proc. of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, Proc. of AAMAS, pages 236–243, Washington, DC, USA, 2004. IEEE Computer Society.
21. J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: An organizational view of multi-agent systems. In *Proc. of AOSE*, pages 214–230. Springer, 2003.
22. C. Giacomo, L. Ferrari, and L. Leonardi. BRAIN: a framework for flexible role-based interactions. In *Proc. of CoopIS*, pages 145–161. Springer, 2003.
23. F. Hammer, A. Derakhshan, Y. Demazeau, and H. H. Lund. A multi-agent approach to social human behaviour in children's play. In *Proc. of IAT, Washington*, pages 403–406, 2006.
24. K. V. Hindriks. Programming rational agents in GOAL. *Multi-Agent Programming: Languages and Tools and Applications*, pages 119–157, 2009.
25. A. Hübner, G. P. Dimuro, A. C. R. Costa, and V. L. D. Mattos. A dialogic dimension for the Moise+ organization model. In *Proc. of MALLOW*, pages 21–26, 2010.
26. J. F. Hübner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.
27. J. F. Hübner, J. S. Sichman, and O. Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *Proc. of SBIA*, pages 118–128, London, UK, 2002. Springer.
28. M. Huget and B. Vitteau. Modularity in interaction protocols. In *Workshop on Agent Communication Languages'03*, pages 291–309, 2003.
29. M. N. Huhns and L. M. Stephens. In G. Weiss, editor, *Multiagent Systems*, chapter Multiagent Systems and Societies of Agents, pages 79–120. MIT Press, Cambridge, MA, USA, 1999.
30. D. Keil and D. Q. Goldin. Indirect interaction in environments for multi-agent systems. In *Proc. of E4MAS*, pages 68–87, 2005.
31. Y. Kubera, P. Mathieu, and S. Picault. Interaction-oriented agent simulations: From theory to implementation. In *Proc. of ECAI*, pages 383–387, Patras, Greece, 2008. IOS Press.
32. T. Miller and P. McBurney. Using constraints and process algebra for specification of first-class agent interaction protocols. In *Proc. of ESAW VII*, pages 245–264, Berlin, Heidelberg, 2007. Springer.
33. T. Miller and P. McBurney. On illegal composition of first-class agent interaction protocols. In *Proc. of ACSE*, pages 127–136, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.
34. T. Miller and J. Mcginnis. Amongst first-class protocols. In *Proc. of ESAW VIII*, pages 208–223. Springer-Verlag, Berlin, Heidelberg, 2008.
35. E. Oliva, M. Viroli, A. Omicini, and P. Mcburney. Argumentation and artifact for dialogue support. In *Argumentation in Multi-Agent Systems*, LNAI, pages 107–121. Springer-Verlag, Berlin, Heidelberg, 2009.
36. A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17:432–456, 2008.
37. S. Paurobally and J. Cunningham. Achieving common interaction protocols in open agent environments. In *Proc. of AAMAS*, 2002.

38. S. Paurobally and J. Cunningham. Developing agent interaction protocols using graphical and logical methodologies. In *Proc. of PROMAS*, pages 149–168. Springer, 2003.
39. E. Platon, N. Sabouret, and S. Honiden. Overhearing and direct interactions: point of view of an active environment, a preliminary study. In *Proc. of E4MAS*, pages 121–138. Springer, 2005.
40. A. Ricci, M. Viroli, and A. Omicini. CArtAgO: An infrastructure for engineering computational environments in MAS. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *Proc. of E4MAS*, pages 102–119, Hakodate, Japan, 2006.
41. T. F. Rodrigues, A. C. da Rocha Costa, and G. P. Dimuro. A communication infrastructure based on artifacts for the JaCaMo platform. In *Proc. of the 1st AAMAS Workshop on Engineering MultiAgent Systems*, pages 97–111, 2013.
42. J. Saunier and F. Balbo. Regulated multi-party communications and context awareness through the environment. *Multiagent Grid Syst.*, pages 75–91, 2009.
43. V. T. Silva, R. Choren, and C. J. P. de Lucena. A UML based approach for modeling and implementing multi-agent systems. In *Proc. of AAMAS*, pages 914–921, Washington, DC, USA, 2004. IEEE Computer Society.
44. M. P. Singh. Information-driven interaction-oriented programming: BSPL, the blindingly simple protocol language. In *Proc. of AAMAS*, pages 491–598, 2011.
45. M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, pages 285–312, 2000.
46. P. Yolum and M. P. Singh. Designing and executing protocols using the event calculus. In *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, pages 27–28. ACM, 2001.
47. P. Yolum and M. P. Singh. Commitment machines. In *Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, ATAL '01, pages 235–247. Springer-Verlag, 2002.
48. P. Yolum and M. P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. In *Annals of Mathematics and Artificial Intelligence*, 2004.
49. M. R. Zatelli and J. F. Hübner. A unified interaction model with agent, organization, and environment. In *Anais do IX ENIA@BRACIS*, Curitiba, Brazil, 2012.
50. M. R. Zatelli and J. F. Hübner. A language to specify the interaction considering agents, environment, and organization. In *Anais do VII WESAAC*, São Paulo, Brazil, 2013.