8

Norm-aware and Norm-oriented Programming

Matteo Baldoni,Cristina Baroglio, Olivier Boissier, Jomi F. Hübner, and Roberto Micalizio

ABSTRACT. This chapter introduces programming with norms from a software engineering perspective. A computational system is conceived as being composed of agents (as process), environment (as data), and norms (as control over actions) as first-class entities. Norms are presented from two points of view: system and agent. The first point of view focuses on how to define the expected behavior of agents by means of normative programs, and how this kind of program is integrated with the environment by constitutive norms. The second focuses on how agents, who cannot bring about some goals on their own, can however engage in norm-aware interactions with expectations about the behavior of one another, even in the absence of organizational guidelines. The JaCaMo platform is used to provide concrete examples and illustrate the approach.

1 Introduction

Multiagent systems involve different abstractions. At the individual level, agents are situated in an environment, in which they act. Quoting [Weyns *et al.*, 2007] "the environment is a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources." At the system level, it is widely recognized that further abstractions become handy, like organizations and interactions [Demazeau, 1995], aimed at enabling a meaningful and fruitful coordination of the autonomous and heterogeneous agents in the system. Thus, agents are not only situated in a physical environment, they are also situated in a social environment [Lindblom and Ziemke, 2003] where they have relationships with other agents and are subject to the regulations of the society they belong to. Lopez and Scott [Lopez and Scott, 2000] indentified two components in social situatedness. On the one hand, an institutional component where the "... social structure is seen as comprising those cultural or normative patterns that define the expectations agents hold about each other's behaviour and that organize their enduring relations with each other." On the other hand, a relational component where social structure amounts to "the relationships themselves, understood as patterns of causal interconnection and interdependence among agents and their actions, as well as the positions that they occupy." Norms, and normative reasoning, are at the basis of both components of social situatedness. Then, at the system level, norms produce obligations that drive the agents' behavior, while at the agent level, commitments oblige agents towards each other.

¡book title¿, 1–19.

Only a few proposals in the literature tackle in an integrated way agents, environment, and norms. In order to explain the value of considering computational systems as based on these three abstractions, this chapter positions such kind of systems in the landscape of modularization of software, ranging from functional decomposition to business artifacts, using as a touchstone Meyer's forces of computation [Meyer, 1997]. Then, it introduces some state-of-the-art tools that integrate agents, environment, and norms. Such tools show the versatility of norms as programming components that can be used to specify how agents are coordinated at a system level, how they act on the environment, how the environment impacts on the normative state, and how agents can autonomously and directly create a coordination of their activities on an agent-to-agent basis.

2 Software Engineering Perspective

In order to highlight the role of norms in system programming, we resort to *Meyer's forces of computation*. They provide a neutral touchstone, unrelated to any specific programming approach or modularization mechanism. According to Meyer, three forces are at play when we use software to perform some computations [Meyer, 1997, Ch. 5, p. 101]: *processors*, *actions*, and *objects*. A processor can be a *process* or a *thread* (we use both the terms processor and process to refer to this force); actions are the *operations* that make the computation; objects are the *data* to which actions are applied.

A software system, in order to execute, uses processes to apply certain actions to certain objects. The form of the actions depends on the considered level of granularity: they can be instructions of a programming language, as well as they can be major steps of a complex algorithm. Moreover, the form of actions conditions the way in which processes operate on objects. Some objects are built by a computation for its own needs and exist only while the computation proceeds; others (e.g., files or databases) are external and may outlive individual computations. In the following, we analyze the most important proposals concerning software modularization, showing how they, sometimes implicitly, give more or less strength to Meyer's forces, and the drawbacks that follow. We, then, devote a special attention to the abstractions of agent, environment, and norm, explaining how altogether they provide a programming paradigm which is much more balanced, with respect to the use of the three forces, than all the previously considered ones.

Top-down functional decomposition puts in the center the notion of process, where a process is seen as implementing a given function. A system is built by stepwise refinement, each refinement step decreases the abstraction of the specification. The approach disregards objects, just considered as data structures that are instrumental to the function specification and internal to processes. Actions are defined only in terms of the instructions provided by the programming language and of other functions built on top of them, into which a process is structured. As a consequence, this approach, though intuitive and suitable to the development of individual algorithms, does not scale up well when data are shared among concurrent processes.

The *Object-Oriented approach* to modularization results from an effort aimed

at showing the limits of the functional approach [Meyer, 1997]. Objects (data) often have a life of their own, independent from the processes that use them. Objects become, then, the fundamental notion of the model. They provide the actions by which–and only by which–it is possible to operate on them (data operations). Objects have a static nature: actions are invoked on objects; the process that invokes an operation is the one that causes the evolution of the object. Thus, there is no decoupling between the use of an object and the management of that object. Moreover, the model does not supply conceptual notions for representing tasks, in particular when concurrency is involved.

The key concept in the *actor model* [Hewitt *et al.*, 1973] (by which *active objects* are largely inspired) is that everything is an actor. Interaction between actors occurs only through direct asynchronous message passing, with no restriction on the order in which messages are received; recipients of messages are identified by opaque addresses. The decoupling between the sender of a message and the communications sent makes it possible both to tackle asynchronous communication and to define actors' control structures as patterns of passing messages. Many authors, such as [Mitchell, 2002; Tasharofi *et al.*, 2013; Neykova and Yoshida, 2014], note that the actor model does not address the issue of *coordination*. Coordination requires the possibility for an actor to have expectations about another actor's behavior, but asynchronous message passing alone gives no means to foresee how a message receiver will behave. For example, in the object-paradigm, methods return the computed results to their callers. In the actor model this is not guaranteed because this simple pattern requires the exchange of two messages: no way for specifying patterns of message exchanges between actors is provided. The lack of such mechanisms hinders the verification of properties of a system of interacting actors. Similar problems are well-known also in the area that studies enterprise application integration [Alonso *et al.*, 2004] and service-oriented computing [Singh and Huhns, 2005], that can be considered as heirs to the actor model. The above problem can better be understood by referring to Meyer's forces. The actor model supports the realization of object/data management processes (these are the internal behaviors of the actors, that rule how the actor evolves), but it does not support the design and the modularization of processes that perform the object use, which would be *external* to the actors.

Business processes have been increasingly adopted by enterprises and organizations to conceptually describe their dynamics, and those of the sociotechnical systems they live in. More specifically, a business process describes how a set of interrelated activities can lead to a precise and measurable result (a product or a service) in response to an *external event* (e.g., a new order) [Weske, 2007]. Among the main advantages of this process-centric view, we have the fact that it enables analysis of an enterprise functioning, it enables comparison of business processes, it enables the study of compliance to norms (e.g., [Governatori, 2010]), and also to identify critical points like bottlenecks by way of simulations. On the negative side, business processes show the same limits as functional decomposition. Specifically, they are typically represented in an activity-centric way; i.e., by emphasizing which flows of activities are acceptable, without providing adequate abstractions to capture the data that

are manipulated along such flows.

The *artifact-centric approach* [Bhattacharya *et al.*, 2007; Cohn and Hull, 2009; Calvanese *et al.*, 2013] counterposes a data-centric vision to the activity-centric vision described above. *Artifacts* are concrete, identifiable, self-describing chunks of information, the basic building blocks by which business models and operations are described. They include an *information model* of the data, and a *lifecycle model*, that contains the key states through which the data evolve, together with their transitions. The lifecycle model is not only used at runtime but also at design time to understand who is responsible of which transitions. On the negative side, like in the case of the actor model, business artifacts disregard the design and the modularization of those processes that operate on them. Moreover, verification problems are much harder to tackle because the explicit presence of data, together with the possibility of incorporating new data from the external environment, makes these systems infinite-state in general [Calvanese *et al.*, 2013].

2.1 Agents, Environments, and Norms

In [Russell and Norvig, 2003; Wooldridge, 2009], *agents* are defined as entities that observe their environment and act upon it so as to achieve their own goals. Two fundamental characteristics of agents are *autonomy* and *situatedness*. Agents are autonomous in the sense that they have a sense-plan-act deliberative cycle, which gives them control of their internal state and behavior; autonomy, in turn, implies proactivity; i.e., the ability of an agent to take action towards the achievement of its (delegated) objectives without being solicited to do so. Agents are situated because they can sense and manipulate the environment in which operate. The environment can be physical or virtual, and is understood by agents in terms of relevant data. The difference between the agent paradigm and the previously cited paradigms is that the agent paradigm introduces two equally important abstractions: the *agent* and the *environment* [Weyns *et al.*, 2007]. Such a dichotomy does not find correspondence in the other models and gives a first-class role to both Meyer's process and object forces. Processes realize algorithms aimed at achieving objectives. This is exactly the gist of the agent abstraction and the rationale behind its proactivity: agents exploit their deliberative cycle (as control flow), possibly together with the key abstractions of belief, desire, and intention (as logic), in order to realize algorithms (i.e., processes), for acting in their environment and pursue their goals[1]. The manifestation of the object force is the environment abstraction. The environment does not exhibit the kind of autonomy explained for agents even when its definition includes a process. Being reactive, rather than active, makes the environment more similar to an actor whose behavior is triggered by the messages it receives, that are all served equally.

Actions are the capabilities agents have to modify their environment. The process force is mapped onto a cycle in which the agent observes the world (updating its beliefs), deliberates which intentions to achieve, plans how to achieve them, and finally executes the plan [Bratman, 1990]. Beliefs and intentions are

[1]Summarizing, objects "do it" for free because they are data, agents are processes and "do it" because it is functional to their objectives.

those components of the process abstraction that create a bridge respectively towards the object/data force (i.e., the environment) and the action force. Beliefs concern the environment. Intentions lead to action [Wooldridge, 2009], meaning that if an agent has an intention, then the expectation is that the agent will make a reasonable attempt to pursue it. In this sense, intentions play a central role in the selection and the execution of action. Consequently, instead of being subordinate to the process force the action force is put in relation to it by means of intentions. This is a difference with respect to functional decomposition (see previous section), where actions are produced as modular component processes (e.g. procedures) by refining a given goal through a top-down strategy.

The action force is better considered by *normative multiagent systems* [Jones and Carmo, 2001; Boella *et al.*, 2007], which take inspiration from mechanisms that are typical of human communities, and have been widely studied in the research area on multiagent systems. According to [Boella *et al.*, 2007] a normative multiagent system is: "a multiagent system together with normative systems in which agents on the one hand can decide whether to follow the explicitly represented norms, and on the other the normative systems specify how and to which extent the agents can modify the norms". The deliberative cycle of agents is affected by the norms and by the obligations these norms generate as a consequence of the agents' actions. Each agent is free to adapt its behavior to (local or coordination) changing conditions, e.g., by re-ranking its goals based on the context or by adopting new goals. Institutions and organizations set the ground for coordination and cooperation among agents. Intuitively, an institution is an organizational structure for coordinating the activities of multiple interacting agents, that typically embodies some rules (norms) that govern participation and interaction. In general, an organization adds to this societal dimension a set of organizational goals and powers to create institutional facts or to modify the norms and obligations of the normative system [Boella and van der Torre, 2004]. Agents, playing one or more roles, must accomplish the organizational goals respecting the norms. Institutions and organizations are, thus, a way to realize functional decomposition in an agent setting.

In the normative multiagent systems domain, several proposals focus on *regulative norms* that, through obligations, permissions, and prohibitions, specify the patterns of actions and interactions agents should adhere to, even though deviations can still occur and have to be properly considered [Jones and Carmo, 2001]. These regulative norms have been combined with *constitutive norms* [Jones and Sergot, 1997; Noriega, 1997; Boella and van der Torre, 2004; Grossi, 2007; Chopra and Singh, 2008; Criado *et al.*, 2013], which support the creation of institutional realities by defining institutional actions that make sense only within the institutions they belong to. A typical example is that of "raising a hand", which counts as "make a bid" in the context of an auction. Institutional actions allow agents to operate within an institution. Citing [Criado *et al.*, 2013], the impact on the agent's deliberative cycle is that agents can "reason about the social consequences of their actions." In this light, going back to Meyer's forces, if agents are abstractions for processes and environments for objects, then *norms* are abstractions of the *action force* because norms model

actions and, thus, condition the way in which processes operate on objects. In fact, norms specify either institutional actions, or the conditions for the use of such actions, consequently regulating the acceptable behavior of the agents in a system. This view is also supported by the fact that norms concern "doing the right thing" rather than "doing what leads to a goal" [Therborn, 2002].

The development of software systems using the agent-oriented approach requires tools that properly integrate the programming of the agents, the environment and the norms (both regulative and constitutive), all of them as first class entities. Although there are many tools to independently program each of these parts, we have few tools that consider their integration. One example is the integration of 2APL (an agent programming language) and 2OPL (an normative programming language) [Dastani, 2008; Dastani *et al.*, 2009; Dastani, 2015]. An example that also considers the environment as a first class entity is JaCaMo [Boissier *et al.*, 2013]. JaCaMo integrates Jason [Bordini *et al.*, 2007] (an agent programming language), CArtAgO [Ricci *et al.*, 2011] (an environment programming language) and Moise [Hübner *et al.*, 2007] (an organization and normative programming language) and is considered in a later section to illustrate normative programming and awareness.

3 Norm-Oriented Programming

This section presents an implemented programming language for norms to further illustrate how the action force is realized by norms in the case of coordination. The coordination of the agents in JaCaMo can be conceived at two levels: goals and actions. In the former, global plans are used to define dependencies between collective goals[2] that agents can commit to. Due to their commitments, agents are then *obliged* to achieve those goals. They are however free to decide which actions they will perform to achieve them. In the latter, interaction protocols are used to define the exact sequence of actions that are expected from participant agents and the resources they use. Once an agent participates in a protocol, from a system perspective, it is *obliged* to perform the expected actions as specified by the protocol. At both levels, norms are used to express obligations and mechanisms are used to monitor whether agents are compliant, since they are autonomous and might violate norms. In the rest of this section, we briefly present a language to program the norms and then focus on the action level to illustrate the language.

3.1 Normative Programming Language

An important feature in modularizing norms in a dynamic MAS is to keep them independent from both the agents and the environment, so that the developer does not need to update the normative program whenever new agents arrive or the environment changes. While the independence from the agents is usually achieved by the notion of role, the independence from the environment is managed via constitutive rules. For instance, a norm like `the auction winner is obliged to pay` can be applied to several agents (those that play the role of `auction winner`) and environments (those that have concrete elements to be

[2]While collective goals are created by the overall system and possibly shared between agents, individual goals are created by the agents.

interpreted as `to pay`).

In the case of JaCaMo, the Normative Programming Language (NPL) has this property [Hübner *et al.*, 2011]. The language is quite simple and based on just two constructs: *obligation* and *regimentation*[3]. With these two primitives, others constructions are possible:

1. Prohibitions are represented either via regimentation or via an obligation for someone else to decide how to handle the situation (e.g., to impose some sanction). For example, consider the norm "it is prohibited to submit a paper with more than 16 pages". In the case of regimentation of this norm, attempts to submit a paper with more than 16 pages will fail (i.e., they will be prevented from taking place). In case this norm is not to be regimented, the designer could handle the prohibition by defining an obligation for another agent as in "when a paper with more than 16 pages is submitted, the chair must decide whether to accept the submission or not".

2. Permissions are defined by omission, as in [Grossi *et al.*, 2007]; that is, if something is not obligatory nor prohibited it is simply permitted.

3. Sanctions are represented as obligations (i.e., some agent is *obliged* to apply the sanction).

NPL norm syntax has the general form:

$$\texttt{norm}\ id\ \texttt{:}\quad \varphi\ \texttt{->}\ \psi$$

where *id* is a unique *identifier* of the norm; φ is a formula that determines the *activation condition* for the norm; and ψ is the *consequence* of the activation of the norm. Two types of norm consequences ψ are possible:

- *fail* – `fail`(r): represents the case where the norm is regimented; the argument r represents the reason for the failure;

- *obl* – `obligation`(a, r, g, d): represents the case where an obligation for some agent a is created. Argument r is the reason for the obligation; g is the formula that represents the obligation itself (either an action or a state of the world that the agent must bring about); and d is the deadline to fulfil the obligation.

A simple example to illustrate the language is given below:

```
// the auction winner has 4 hours to pay the product
norm n1: winner(A) & bid(A,V)
 -> obligation(A,n1,pay(V),‘now‘+‘4 hours‘).

// example of a regimented norm; bids should be greater than zero
norm n2: bid(_,V) & V <= 0
 -> fail(n2(bid(V))).
```

[3]Regimentation is a preventive strategy to enforce norms whereby agents are not capable of violation [Jones and Sergot, 1993].

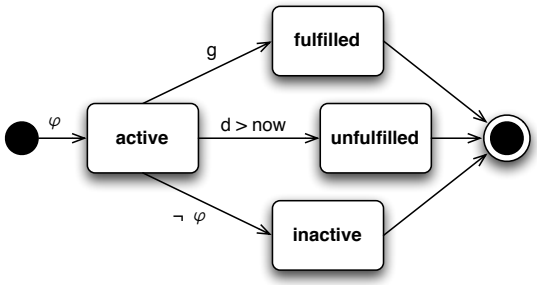

Figure 1. Life-cycle of obligations in NPL

The interpretation of such programs creates obligations for the participating agents. An obligation has a run-time life-cycle as defined in Figure 1 and explained below (the formal semantics is presented in [Hübner *et al.*, 2011]).

1. An obligation is created when the activation condition φ of some norm n holds. The activation condition formula is used to instantiate the values of variables a, r, g, and d of the obligation to be created.
2. Once created, the initial state of an obligation is *active*.
3. The state changes to *fulfilled* when agent a fulfills the norm's obligation g before the deadline d.
4. The obligation state changes to *unfulfilled* when agent a does not fulfil the norm's obligation g before the deadline d.
5. As soon as the activation condition (φ) of the norm that created the obligation ceases to hold, the state changes to *inactive*.

The evaluation of the formulas φ and g are based on institutional facts and actions (as introduced in Sec. 2.1). For instance the fact that Alice has transferred some money to a bank account is considered as satisfying a norm only if this action is institutional. In the case of JaCaMo, all institutional facts and actions are defined by a *count-as* program as proposed in [de Brito *et al.*, 2015; de Brito *et al.*, 2017]. A count-as program links the environment and the institution by defining how concrete facts and actions in the environment are interpreted from an institutional point of view. For instance the following count-as program defines the institutional facts and actions used in the norms presented earlier in the auction scenario.

```
tell(A,bid(V)) count-as bid(A,V)
            // tell is a speech act, a brute fact in the environment
            // and bid/2 is an institutional fact

A count-as winner(A)
  while bid(A,V) & not (bid(A2,V2) & V2 > V1 & A2 \= A1) &
        auction(closed)
            // A is the name of concrete agent
            // and winner/1 is an institutional fact
```

```
 1. protocol voting {
 2.   participants:
 3.     initiator  agent;
 4.     voter      agent;
 5.     ballotBox  artifact;
 6.   states: k1 initial; k2; k3; k4 final;
 7.   transitions:
 8.     k1 - k2 # initiator -- message[tell] "object(X)" -> voter;
 9.     k2 - k3 # voter     -- action "vote(Y)" -> ballotBox
10.     k2 - k3 # timeout 30000;
11.     k3 - k4 # ballotBox -- event "winner(Y)" -> initiator;
12. }
```

Figure 2. Example of a Simple Voting Protocol

```
bank_deposit(A,V) count-as pay(V)
  if winner(A)
            // bank_deposit is a concrete action in the environment
            // pay/1 is an institutional action
            // the back deposit is considered by the institution only if
            // performed by the winner
```

3.2 Interaction Protocols

Interaction protocols are a common tool to define how the agents should behave at the action level. In this section we adopt the proposal presented in [Zatelli and Hübner, 2014; Zatelli *et al.*, 2016] since it has the following main property: it is integrated with both the environment and norms. Regarding the integration with the environment, the protocol language considers both the agent-to-artifact interaction and the agent-to-agent interaction based on speech actions. Regarding the integration with norms, a protocol specification is translated to a set of norms for its regulation.

The Zatelli et al. proposal considers an interaction protocol as composed of a set of participants (agents or artifacts), transitions, and states. Each transition links a source state to a target state and it can be fired by concrete facts from the environment like messages sent, events produced by artifacts, and the execution of actions on artifacts. When some transition is fired, a new state is achieved and the protocol execution evolves. For example, the protocol in Fig. 2 specifies the coordination required for a very simple voting process where agents and artifacts are participating. By this protocol (line 8) we expect a behavior where an `initiator` agent announces (by a `tell` message) the object of the election to `voters` participating in the protocol. Voter should then perform the `vote` action in a ballot artifact (line 9). When all votes have being performed or some timeout has achieved (i.e., the state `k3` is achieved), the ballot artifact produces an event announcing the winner (line 11).

Agents are free to join a particular a protocol, but when they do so (by indicating which kind of participant they will play), the system managing the execution of the protocol will produce obligations for them. These obligations are based on NPL norms automatically created from the protocol specification. The creation of these norms is quite simple: one norm is created for each

transition where the source is an agent.[4] For instance, the norms for the protocol of Fig. 2 are:

```
norm k1_k2_a: state(k1) & play(A,initiator) & play(B,voter)
-> obligation(A, voting_protocol, send(A,B,tell,object(_)), 'now'+'1 hour')

norm k2_k3_a: state(k2) & play(A,voter) & play(B,ballotBox)
-> obligation(A, voting_protocol, do(A,B,vote(_)), 'now'+'1 hour')
```

Most of the institutional facts used by these norms are produced by count-as rules: `play(A,P)` is based on the action of agent `A` joining the protocol as participant `P`; `send(A,B,P,C)` is based on the agent `A` sending a `P` message to `B` with content that match `C`; and `do(A,B,O)` is based on the agent `A` doing the action `O` on artifact `B`. However, the institutional fact `state(S)` is internally produced by the protocol management system. For instance, when the obligation of the initiator is fulfilled by sending the voting object message, the protocol evolves from state `k1` to `k2` and thus the institutional fact `state(k2)` holds.

An agent playing voter is obliged to vote not when it joins the protocol, but *after* receiving the message with the voting object, since the norm `k2_k3_a` that obliges it to vote is activated if the protocol is in state `k2`.

By means of norms derived from a protocol specification, we are thus regulating and coordinating the agent at the *action* level. Of course it is a design decision to regulate the agents at that level. For some application it is preferable to regulate and coordinate the agents with norms referring to goals, letting agents to select the proper actions to achieve them.

This section has focused on the norms independently of how the agents will handle them: obligations are created by the normative system towards the agents, setting the expected behavior from a system perspective. The system has monitoring mechanisms to verify whether the agents are following the norms, despite their internal architectures. Nevertheless, better results can be achieved if agents are able to reason about norms. The next section explores how the agents can internally handle what is expected from them to do.

4 Norm-aware Interaction

Since the late '80s, studies on distributed artificial intelligence, studies on formal theories of collective activity, team, or group work, and studies on cooperation implicitly identified in *commitment* the glue of group actvity: commitments link the actions of the group members and the group members with each other [Castelfranchi, 1995; Singh, 1997; Norman *et al.*, 2003]. In particular, *social commitments* [Singh, 1999] are a kind of social relationship with a normative value, that makes it possible for the agents to have expectations on one another and coordinate their activities. On this foundation, works like [Baldoni *et al.*, 2015b; Baldoni *et al.*, 2015a; Baldoni *et al.*, 2018a] propose to complement the interaction protocol in [Zatelli and Hübner, 2014], and more in general organizational approaches, with a relational representation of interaction, where agents, by their own action, directly create normative bonds

[4] Timeout transitions and those triggered by artifacts do not require regulation by norms, since there is no autonomy involved in the transition.

(represented by social commitments) with one another, and use such bonds to coordinate their activities.

A social commitment models the directed relation between two agents: a *debtor* and a *creditor*, that are both aware of the existence of such a relation and of its current state: A commitment $C(x, y, s, u)$ captures that agent x (debtor) commits to agent y (creditor) to bring about the consequent condition u when the antecedent condition s holds. Antecedent and consequent conditions are conjunctions or disjunctions of events and commitments. Since debtors are expected to satisfy their engagements, commitments have a normative value, providing social expectations on the agents' behaviors.

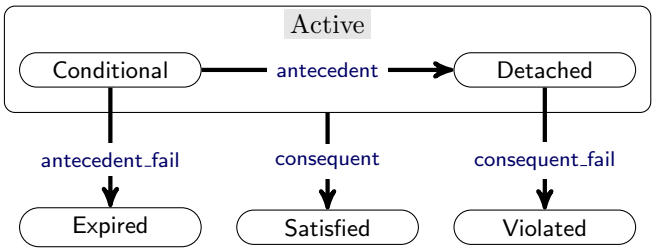

Figure 3. Commitment life cycle.

A commitment is autonomously taken by a debtor towards a creditor on its own initiative and is manipulated by agents through the standard operations *create*, *cancel*, *release*, *discharge*, *assign*, *delegate* [Singh, 1999]. Commitment evolution follows the lifecycle formalized in [Telang *et al.*, 2011], which is reported in Figure 3. A commitment should be *Active* when it is initially created. Active has two substates: *Conditional* as long as the antecedent does not occur, and *Detached* when the antecedent has occurred. A commitment is *Violated* either when its antecedent is true but its consequent will forever be false, or when it is canceled when Detached. It is *Satisfied*, when the engagement is accomplished. It is *Expired*, when it is no longer in effect and therefore the debtor would not fail to comply even if does not accomplish the consequent.

A social commitment, whose antecedent condition is true, amounts to a directed obligation with an important difference. In essence, an obligation is a system level norm while a commitment is an agent level constraint. At system level, something happens and an obligation is created on some agent. At the agent level, an agent creates a conditional social commitment towards some other agent, based on its own beliefs and goals [Telang *et al.*, 2012]. In the most typical case, such a commitment binds the debtor agent to bring about the consequent condition, in the context in which the antecedent condition holds.[5] The creditor agent will detach the conditional commitment if and when it deems it useful to its own purposes, thus activating the obligation of the debtor agent. The motive that leads the two agents to behave in this way is that they have goals they are not able to achieve on their own, so they seek for

[5] In general the debtor is not requested to make the condition true by its own actions but will be liable in case of violation.

cooperation by way of interaction. Such agents do so also because they do not have behavioral guidelines, provided by an organization. The interaction, i.e. the causal relationship between the actions of the two concerned agents, is an effect of the presence of the conditional commitment. The creditor performs some kind of normative reasoning on such a commitment, inferring how to act in order to activate the obligation for its debtor to make the consequent condition true.

The choice of commitments is, thus, motivated by the fact that they are taken by an agent as a result of an internal deliberative process. This preserves the autonomy of the agents and is fundamental to harmonize deliberation with goal achievement. The agent does not just react to some obligations, but it rather includes a deliberative capacity by which it creates engagements towards other agents while it is trying to achieve its goals (or to the aim of achieving its goals). Citing Singh [Singh, 2011], an agent would become a debtor of a commitment based on the agent's own communications: either by directly saying something or having another agent communicate something in conjunction with a prior communication of the debtor. That is, there is a causal path from the establishment of a commitment to prior communications by the debtor of that commitment.

Commitment-based interaction protocols assume that a (notional) *social state* is available and inspectable by all the involved agents [Baldoni *et al.*, 2015c]. The social state traces which commitments currently exist between any two agents, and the states of these commitments according to the commitments life-cycle. The explicit representation of the life-cycle of commitments can be used as an interface towards agent behaviors, in order to tackle those state transitions that are of interest for the achievement of the agent's goals. The social state is, thus, a concrete piece of information that belongs to the environment (object force), and that, by evolving according to a known lifecycle, recalls the business artifacts as defined by data-centric approaches [Nigam and Caswell, 2003; Cohn and Hull, 2009; Baldoni *et al.*, 2016]. The structure and lifecycle of such a piece of information are pivotal in harmonizing the object force with the action force through the commitments, and offer to the agents a precious element that can be used both at design time (for programming the agents), and at run-time to allow agents to take into account also the current commitments and their expected evolution in the process of deciding how to operate. Agents will act upon the social state to achieve their goals by creating new commitments or by detaching/discharging the currently active commitments.

Commitments can, thus, be used by agents in their practical reasoning together with beliefs, intentions, and *goals*. In particular, Telang et al. [Telang *et al.*, 2012] point out that goals and commitments are complementary: a commitment specifies how an agent relates to another one, and hence describes what an agent is willing to bring about for another agent. On the other hand, a goal denotes an agent's attitude towards some condition; that is, a state of the world that the agent should achieve. An agent can create a commitment towards another agent to achieve one of its goals; but at the same time, an agent determines the goals to be pursued relying on the commitments it has

towards others.

4.1 Programming Interaction with Commitments

JaCaMo+ is an extension of JaCaMo that allows Jason agents to engage commitment-based interactions which are reified as CArtAgO artifacts. In JaCaMo+ an artifact represents the social state of an interaction and provides the roles agents enact. The use of artifacts enables the implementation of monitoring functionalities for verifying that the on-going interactions respect the commitments and for detecting violations and violators. Specifically, a JaCaMo+ artifact encodes a commitment protocol, that is structured into a set of roles. By enacting a role, an agent gains the rights to perform social actions, whose execution has public social consequences, expressed in terms of commitments. If an agent tries to perform an action which is not associated with the role it is enacting, the artifact raises an exception that is notified to the violator. On the other hand, when an agent performs an action that pertains to its role, the social state is updated accordingly by adding new commitments, or by modifying the state of existing commitments.

JaCaMo+ extends the *Jason* component of JaCaMo by allowing the specification of plans whose triggering events involve commitments. JaCaMo+ represents a commitment as a term $cc(debtor, creditor, antecedent, consequent, status)$ where *debtor* and *creditor* identify the involved agents (or agent roles), while *antecedent* and *consequent* are the commitment conditions. *Status* is the commitment state (the set being defined in the commitments life-cycle). Commitments operations (e.g. create) are realized as CArtAgO internal operations. Thus, commitment operations cannot be invoked directly by the agents, but the commitment protocol actions will use them as primitives to modify the social state. In a Jason plan specification, commitments can be used wherever beliefs can be used. In contrast to beliefs, their assertion/deletion can only occur through the artifact, in consequence to a social state change. This template shows a Jason plan triggered by the addition of a commitment in the social state:

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

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

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

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

the desired social relationship. After the execution of the plan, the commitment $cc(debtor, creditor, antecedent, consequent, status)$ will hold in the social state, and will be projected onto the belief bases of all agents focusing on the artifact.

Let us see how Dijktra's dining philosophers can be programmed in JaCaMo+. Intuitively, this example shows the advantage of a separation of concerns between the coordination logic in terms of norms (i.e., the action force, as explained in Section 2.1) and the agent's logic (i.e., the process force). When these two forces are kept separate, a better software modularization is possible. In fact, on one hand, it becomes possible to implement and verify the interaction artifact independently of the agents that will use it. On the other hand, it becomes possible to implement the agents' plans in different ways as long as they keep on addressing the commitments' state changes that may occur along the interaction.

```
1  counter(0).
2  !start.
3  +!start: true
4       <- focusWhenAvailable("philoArtifact"); enact("philosopher").
5  +enacted(Id,"philosopher",Role_Id)
6       <- +enactment_id(Role_Id); .my_name(Me);
7          in("philo_init",Me,Left,Right);
8          +my_left_fork(Left); +my_right_fork(Right); !!living.
9  +!living:
10      <- !thinking; !eating.
11 +!eating:my_left_fork(Left) & my_right_fork(Right) & counter(C)
12    <- ?enactment_id(Role_Id);
13       askForks(Left, Right, C).
14 +cc(My_Role_Id, "philosopher", available(Left,Right,C),
15       returnForks(Left,Right,C),"DETACHED")
16    :  enactment_id(My_Role_Id) & my_left_fork(Left) &
17       my_right_fork(Right) & counter(C)
18    <- !eat(Left, Right, C); returnForks(Left, Right, C).
19 +cc(My_Role_Id, "philosopher", available(Left,Right,C),
20      returnForks(Left,Right,C), "SATISFIED")
21    :  enactment_id(My_Role_Id) & my_left_fork(Left)
22    <- ?counter(C); -+counter(C+1); !living.
23 +!eat(Left, Right, C): my_left_fork(Left) & my_right_fork(Right)
24       & available(Left, Right, C) & counter(C)
25    <- .my_name(Me); ?enactment_id(Role_Id);
26       println(Me, " ", Role_Id, " eating").
27 +!thinking: counter(C)
28      <- .my_name(Me); ?enactment_id(Role_Id);
29         println(Me, " ", Role_Id, " thinking, time ",C).
```

Listing 1.1. The philosopher agent program in JaCaMo+.

An agent has a !*living* main cycle (ln. 9) that alternates the goals !*thinking* and !*eating*. Coordination is needed just for eating: to this aim, forks must be available. The interaction artifact provides the role *philosopher*, empowered with an operation *askForks*. When an agent wants to eat, it invokes such an operation which in turn creates a commitment to return the forks, whose antecedent condition is to have the forks assigned. Eventually, forks will be ready and the commitment will be detached, so the agent can use the forks (i.e., eat) before discharging the commitment by returning them. The agent who executes the operation is the debtor of such a commitment, any other philosopher is the creditor. The antecedent condition is that forks are available and the consequent is that forks will be returned. Note that fork assignment is decided by way of a coordination policy that is implemented in the artifact.

The *askForks* operation hides the synchronization for using forks. The only concern on the agent side is to address the meaningful state changes of the commitment that was created by means of *askForks*. In our case, only *Detached* and *Satisfied* are meaningful. When the commitment is detached, the agent eats and then executes *returnForks*, thus satisfying its commitment. When the commitment is satisfied, the agent can re-start its main cycle (!*living*). Knowing the social meanings of artifact operations is sufficient for coordinating with others correctly. The connection between the event "commitment detached" and the associated plan is not only causal, but rather the plan has the aim of satisfying the consequent condition of the commitment (*returnForks*).

5 Challenges

The development of software systems with integrated agent, environment, and norms can still be improved in several aspects. We mention here three challenges related to the environment dimension:

- While the impact of the environment on the normative state is addressed by the constitutive rules, the proper design and instrumentalization of the environment to achieve organizational goals still deserves further investigation.

- Agent coordination should not only concern agent activities but it should also account for, and in some cases be driven by, the environment and its evolution. One major challenge is that this kind of coordination calls for a declarative specification of the environment life cycle, upon which the normative system should be based. The advantage would be twofold. Agents would be capable of reasoning on the system as a whole (because they would have expectations on both other agents and on the environment) with an understanding of the implications on their own behavior, in terms of duties, prohibitions and such like. On the other hand, it would also be possible to perform property analysis at the level of specification and norms rather than on the system as a whole.

- A third challenge concerns tackling accountability through a proper formalization and to support agent programming through typing. Accountability is a fundamental concept at the basis of interaction which is still little explored in the MAS literature [Chopra and Singh, 2016; Baldoni *et al.*, 2018c]. It concerns the identification of who should give account for some situation of interest. In organizational contexts it is often used when some undesired situation occurs, with the aim of improving performance. Typing concerns capturing those requirements that agent programs should satisfy, for instance, to play an organizational role [Baldoni *et al.*, 2018b]. It is a feature at the heart of software engineering.

BIBLIOGRAPHY

[Alonso *et al.*, 2004] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services*. Springer, 2004.

[Baldoni *et al.*, 2015a] Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. Empowering Agent Coordination with Social Engagement. In M. Gavanelli, E. Lamma, and F. Riguzzi, editors, *AI*IA 2015: Advances in Artificial Intelligence, XIV International Conference of the Italian Association for Artificial Intelligence*, volume 9336 of *LNAI*, pages 89–101, Ferrara, Italy, September 2015. Springer.

[Baldoni *et al.*, 2015b] Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. Exploiting Social Commitments in Programming Agent Interaction. In Q. Chen, P. Torroni, S. Villata, J. Y. Hsu, and A. Omicini, editors, *PRIMA 2015: Principles and Practice of Multi-Agent Systems, 18th International Conference*, number 9387 in Lecture Notes in Computer Science, pages 566–574, Bertinoro, Italy, October 26th–30th 2015. Springer.

[Baldoni *et al.*, 2015c] Matteo Baldoni, Cristina Baroglio, Amit K. Chopra, and Munindar P. Singh. Composing and Verifying Commitment-Based Multiagent Protocols. In Q. Yang and M. Wooldridge, editors, *Proc. of 24th International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 10–17, Buenos Aires, Argentina, July 25th-31th 2015. AAAI Press / International Joint Conferences on Artificial Intelligence.

[Baldoni *et al.*, 2016] Matteo Baldoni, Cristina Baroglio, Diego Calvanese, Roberto Micalizio, and Marco Montali. Towards Data- and Norm-Aware Multiagent Systems. In *Engineering Multi-Agent Systems - 4th International Workshop, EMAS 2016, Singapore, Singapore, May 9-10, 2016, Revised, Selected, and Invited Papers*, pages 22–38, 2016.

[Baldoni *et al.*, 2018a] Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. Commitment-based Agent Interaction in JaCaMo+. *Fundamenta Informaticae*, 159:1–33, 2018.

[Baldoni *et al.*, 2018b] Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. Type Checking for Protocol Role Enactments via Commitments. *Journal of Autonomous Agents and Multi-Agent Systems*, 2018. In press, DOI: https://doi.org/10.1007/s10458-018-9382-3.

[Baldoni *et al.*, 2018c] Matteo Baldoni, Cristina Baroglio, Katherine M. May, Roberto Micalizio, and Stefano Tedeschi. Computational Accountability in MAS Organizations with ADOPT. *Journal of Applied Sciences, special issue "Multi-Agent Systems"*, 8(4):489, March 2018.

[Bhattacharya *et al.*, 2007] Kamal Bhattacharya, Nathan S. Caswell, Santhosh Kumaran, Anil Nigam, and Frederick Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.

[Boella and van der Torre, 2004] Guido Boella and Leendert W. N. van der Torre. Regulative and constitutive norms in normative multiagent systems. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*, pages 255–266. AAAI Press, 2004.

[Boella *et al.*, 2007] Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen. Introduction to normative multiagent systems. In Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen, editors, *Normative Multi-agent Systems, 18.03. - 23.03.2007*, volume 07122 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[Boissier *et al.*, 2013] Olivier Boissier, Rafael H. Bordini, Jomi F. Hb̈ner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747 – 761, 2013.

[Bordini *et al.*, 2007] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

[Bratman, 1990] Michael E. Bratman. What is intention? In P. Cohen, J. Morgan, and M. Pollack, editors, *Intensions in Communication*, pages 15–31. MIT Press, Cambridge, MA, 1990.

[Calvanese *et al.*, 2013] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. Foundations of data-aware process analysis: a database theory perspective. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 1–12. ACM, 2013.

[Castelfranchi, 1995] Cristiano Castelfranchi. Commitments: From Individual Intentions to Groups and Organizations. In Victor R. Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multiagent Systems (ICMAS)*, pages 41–48, San Francisco, California, USA, June 1995. The MIT Press.

[Chopra and Singh, 2008] Amit K. Chopra and Munindar P. Singh. Constitutive interoperability. In *Proceedings of the 7th Int. J. Conf. on Autonomous agents and multiagent systems, Volume 2*, pages 797–804. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[Chopra and Singh, 2016] Amit K. Chopra and Munindar P. Singh. From social machines to social protocols: Software engineering foundations for sociotechnical systems. In *Proc. of the 25th Int. Conf. on WWW*, 2016.

[Cohn and Hull, 2009] David Cohn and Richard Hull. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.

[Criado *et al.*, 2013] Natalia Criado, Estefania Argente, Pablo Noriega, and Vicent Botti. Reasoning about constitutive norms in BDI agents. *Logic Journal of IGPL*, 2013.

[Dastani *et al.*, 2009] Mehdi Dastani, Nick A. M. Tinnemeier, and John-Jules Ch. Meyer. A programming language for normative multi-agent systems. In V. Dignum, editor, *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, chapter XVI, pages 397–417. Information Science Reference, Hershey, PA, USA, 2009.

[Dastani, 2008] Mehdi Dastani. 2APL: a practical agent programming language. *Autonomous Agent and Multi-Agent Systems*, 16:241–248, 2008.

[Dastani, 2015] Mehdi Dastani. Programming multi-agent systems. *Knowledge Eng. Review*, 30(4):394–418, 2015.

[de Brito *et al.*, 2015] Maiquel de Brito, Jomi F. Hübner, and Olivier Boissier. Bringing constitutive dynamics to situated artificial institutions. In *Proc. of 17th Portuguese Conference on Artificial Intelligence (EPIA 2015)*, volume 9273 of *LNCS*, pages 624–637. Springer, 2015.

[de Brito *et al.*, 2017] Maiquel de Brito, Jomi Fred Hübner, and Olivier Boissier. Situated artificial institutions: stability, consistency, and flexibility in the regulation of agent societies. *Autonomous Agents and Multi-Agent Systems*, pages 1–33, 2017.

[Demazeau, 1995] Yves Demazeau. From interactions to collective behaviour in agent-based systems. In *Proceedings of the 1st. European Conference on Cognitive Science*, pages 117–132, Saint-Malo, 1995.

[Governatori, 2010] Guido Governatori. Law, logic and business processes. In *Third International Workshop on Requirements Engineering and Law, RELAW 2010, Sydney, NSW, Australia, September 28, 2010*, pages 1–10. IEEE, 2010.

[Grossi *et al.*, 2007] Davide Grossi, Huib Aldewered, and Frank Dignum. *Ubi Lex, Ibi Poena*: Designing norm enforcement in e-institutions. In P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Matson, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, volume 4386 of *LNAI*, pages 101–114. Springer, 2007. Revised Selected Papers.

[Grossi, 2007] Davide Grossi. *Designing Invisible Handcuffs, Formal Investigations in Institutions and Organizations for Multi-agent Systems*. PhD thesis, University of Utrecht, 2007.

[Hewitt *et al.*, 1973] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular ACTOR formalism for artificial intelligence. In Nils J. Nilsson, editor, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, August 1973*, pages 235–245. William Kaufmann, 1973.

[Hübner *et al.*, 2007] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Developing organised multi-agent systems using the MOISE+ model: Programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.

[Hübner *et al.*, 2011] Jomi F. Hübner, Olivier Boissier, and Rafael H. Bordini. A normative programming language for multi-agent organisations. *Annals of Mathematics and Artificial Intelligence*, 62(1-2):27–53, 2011.

[Jones and Carmo, 2001] Andrew J.I. Jones and José Carmo. Deontic logic and contrary-to-duties. In Dov Gabbay, editor, *Handbook of Philosophical Logic*, pages 203–279. Kluwer, 2001.

[Jones and Sergot, 1993] Andrew J. I. Jones and Marek Sergot. On the characterization of law and computer systems: the normative systems perspective. In *Deontic logic in computer science: normative system specification*, pages 275–307. John Wiley and Sons Ltd., Chichester, UK, 1993.

[Jones and Sergot, 1997] Andrew J. I. Jones and Marek J. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4:429–445, 1997.

[Lindblom and Ziemke, 2003] Jessica Lindblom and Tom Ziemke. Social Situatedness of Natural and Artificial Intelligence: Vygotsky and Beyond. *Adaptive Behaviour*, 11(2):79–96, 2003.

[Lopez and Scott, 2000] Jose Lopez and John Scott. *Social Structure*. Open University Press, 2000.

[Meyer, 1997] Bertrand Meyer. *Object-oriented Software Construction (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[Mitchell, 2002] John C. Mitchell. *Concepts in programming languages*. Cambridge University Press, Cambridge, New York (N. Y.), 2002.

[Neykova and Yoshida, 2014] Rumyana Neykova and Nobuko Yoshida. Multiparty Session Actors. In eva Kühn and Rosario Pugliese, editors, *Coordination Models and Languages - 16th IFIP WG 6.1 International Conference, COORDINATION 2014, Held as Part of the 9th International Federated Conferences on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings*, volume 8459 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2014.

[Nigam and Caswell, 2003] Anil Nigam and Nathan S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[Noriega, 1997] Pablo Noriega. *Agent-Mediated Auctions: The Fishmarket Metaphor*. PhD thesis, Institut d'Investigatió en Intelligència Artificial, 1997.

[Norman *et al.*, 2003] Timothy J. Norman, D. V. Carbogim, Eric C. W. Krabbe, and C. Douglas Walton. Argument and multi-agent systems. In *Argumentation Machines: New Frontiers in Argument and Computation, volume 9 of Argumentation Library*, pages 15–54. 2003.

[Ricci *et al.*, 2011] Alessandro Ricci, Michele Piunti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.

[Russell and Norvig, 2003] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[Singh and Huhns, 2005] Munindar P. Singh and Michael N. Huhns. *Service-oriented computing - semantics, processes, agents*. Wiley, 2005.

[Singh, 1997] Munindar P. Singh. Commitments Among Autonomous Agents in Information-Rich Environments. In *Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent Rationality*, pages 141–155, London, UK, UK, 1997. Springer-Verlag.

[Singh, 1999] Munindar P. Singh. An ontology for commitments in multiagent systems. *Artif. Intell. Law*, 7(1):97–113, 1999.

[Singh, 2011] Munindar P. Singh. Commitments in multiagent systems some controversies, some prospects. In Fabio Paglieri, Luca Tummolini, Rino Falcone, and Maria Miceli, editors, *The Goals of Cognition. Essays in Honor of Cristiano Castelfranchi*, chapter 31, pages 601–626. College Publications, London, 2011.

[Tasharofi *et al.*, 2013] Samira Tasharofi, Peter Dinges, and Ralph E. Johnson. Why Do Scala Developers Mix the Actor Model with Other Concurrency Models? In *Proceedings of the 27th European Conference on Object-Oriented Programming*, ECOOP'13, pages 302–326, Berlin, Heidelberg, 2013. Springer-Verlag.

[Telang *et al.*, 2011] Pankaj R. Telang, Munindar P. Singh, and Neil Yorke-Smith. Relating goal and commitment semantics. In *ProMAS*, volume 7217 of *Lecture Notes in Computer Science*, pages 22–37. Springer, 2011.

[Telang *et al.*, 2012] Pankaj R. Telang, Neil Yorke-Smith, and Munindar P. Singh. Relating Goal and Commitment Semantics. In *Proc. of ProMAS*, volume 7212 of *LNCS*, pages 22–37. Springer, 2012.

[Therborn, 2002] Göran Therborn. Back to norms! on the scope and dynamics of norms and normative action. *Current Sociology*, 50:863–880, 2002.

[Weske, 2007] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

[Weyns *et al.*, 2007] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *JAAMAS*, 14(1):5–30, 2007.

[Wooldridge, 2009] Michael J. Wooldridge. *Introduction to multiagent systems, 2nd edition*. Wiley, 2009.

[Zatelli and Hübner, 2014] Maicon R. Zatelli and Jomi F. Hübner. The interaction as an integration component for the JaCaMo platform. In Fabiano Dalpiaz, Jürgen Dix, and M. Birna van Riemsdijk, editors, *Proc. 2nd International Workshop on Engineering Multi-agent Systems (EMAS @ AAMAS 2014)*, volume 8758 of *LNCS*, pages 431–450. Springer, 2014.

[Zatelli *et al.*, 2016] Maicon Rafael Zatelli, Alessandro Ricci, and Jomi F. Hb̈ner. Integrating interaction with agents, environment, and organisation in JaCaMo. *International Journal of Agent-Oriented Software Engineering (IJAOSE)*, 5(2,3):266 – 302, 2016.

Matteo Baldoni
Università degli Studi di Torino
Dipartimento di Informatica
via Pessinetto 12, I10149, Torino (TO) Italy
Email: matteo.baldoni@unito.it

Cristina Baroglio
Università degli Studi di Torino
Dipartimento di Informatica
via Pessinetto 12, I10149, Torino (TO) Italy
Email: cristina.baroglio@unito.it

Olivier Boissier
address Mines Saint-Etienne
158 cours Fauriel
42023 Saint-Etienne, France
Email: Olivier.Boissier@emse.fr

Jomi F. Hübner
Federal University of Santa Catarina
Department of Automation and Systems Engineering
PO Box 476, Florianpolis, SC, 88040-900 Brazil
Email: jomi.hubner@ufsc.br

Roberto Micalizio
Università degli Studi di Torino
Dipartimento di Informatica
via Pessinetto 12, I10149, Torino (TO) Italy
Email: roberto.micalizio@unito.it