applied sciences

MDPI

Article

Computational Accountability in MAS Organizations with ADOPT

Matteo Baldoni *,†, Cristina Baroglio †, Katherine M. May †, Roberto Micalizio † and Stefano Tedeschi †

Dipartimento di Informatica, Università degli Studi di Torino, via Pessinetto 12, I10149 Torino, Italy; baroglio@di.unito.it (C.B.); katherine.may@edu.unito.it (K.M.M.); roberto.micalizio@unito.it (R.M.); tedeschi@di.unito.it (S.T.)

* Correspondence: baldoni@di.unito.it; Tel.: +39-011-670-6756

† These authors contributed equally to this work.

Received: 28 February 2018; Accepted: 20 March 2018; Published: 23 March 2018

Abstract: This work studies how the notion of accountability can play a key role in the design and realization of distributed systems that are open and that involve autonomous agents that should harmonize their own goals with the organizational goals. The socio–technical systems that support the work inside human companies and organizations are examples of such systems. The approach that is proposed in order to pursue this purpose is set in the context of multiagent systems organizations, and relies on an explicit specification of relationships among the involved agents for capturing who is accountable to whom and for what. Such accountability relationships are created along with the agents' operations and interactions in a shared environment. In order to guarantee accountability as a design property of the system, a specific interaction protocol is suggested. Properties of this protocol are verified, and a case study is provided consisting of an actual implementation. Finally, we discuss the impact on real-world application domains and trace possible evolutions of the proposal.

Keywords: methodologies for agent-based systems; organizations and institutions; socio–technical systems; computational accountability; social commitments; agent-based programming

1. Introduction

The design of complex distributed systems, such as socio–technical systems (STSs), requires the coordination of activities that are carried out simultaneously by different software components, that is, the interfaces through which humans interact. Multiagent systems (MASs) allow tackling this problem by providing modeling, like [1–5], development approaches, like [6,7], and frameworks, like [8–11]. Broadly speaking, such solutions represent software components as goal-oriented, autonomous agents, which act in a shared environment and need to coordinate so as to achieve their goals [12].

This goal-oriented approach, in which modularity is realized through the assignment of subgoals to the agents, is criticized in [13–15] because it does not fit the realization of open systems, which comprise autonomous agents, each with their own goals to accommodate in a greater picture. For instance, an agent could be assigned a goal it has no capability to achieve, making the whole system vulnerable. Interaction and a representation of the responsibilities that are explicitly taken by the agents are suggested by those works to play a central role. The proposal that we present in this paper develops this perspective (i) by supplying a definition of computational accountability [14,16] and (ii) by providing both modeling and computational tools that can be used in actual implementations. Accountability is a well-known key resource inside human organizations: it fosters the creation of commitments that will help the organization to meet expectations, develop and align strategies, assign resources, analyze and react to failures, and adapt processes to evolving environmental

conditions. In decision making, accountability constrains the set of acceptable alternatives, and thus it allows organization members to take decisions in a way that goes beyond the private beliefs, goals and psychological dispositions of the decision maker [17]. At the core of the proposal lies an interpretation of accountability as a design property. We will explain how it is possible to design systems where accountability is a property that is guaranteed by design, and where the monitoring of ongoing interactions will allow identifying behaviors that diverge from the expected. The proposed solution builds upon the equally important notions of control and of accountability relationship. In particular, the realization of computational accountability follows principles, explained in Section 3, that find a realization in the ADOPT (accountability-driven organization programming technique) protocol for creating and manipulating accountability relationships. Technically, the core of the proposal builds upon the notion of role and in the action of role adoption (or enactment), on one side, and on the concept of social commitment [18,19] on the other side. An early version of ADOPT was presented in [16]. In this paper we enhance the protocol by clearly separating the role adoption phase from the goal agreement phase, and by relying on well-known FIPA protocols to capture the message exchanges which create the accountability relationships (represented by social commitments), and make them evolve. Moreover, we present, as a case study, an extension of the JaCaMo framework [11] that implements the proposal.

From a practical perspective, the proposal can find a natural application in supporting self-regulatory initiatives in human organizations. For instance, many organizations and companies voluntarily adopt monitoring and accountability frameworks, for example [20,21], but such frameworks are currently very-little supported by software and information systems. The commitments that involve the parties are basically hand-written by filling in forms [22,23], and the assessment of satisfaction or violation of the involved liabilities, as well as the actual accounting process, are totally handled by authorized human parties [24]. The problem is that when accountability channels are informal or ambiguous, in the long run, all these processes that are at the heart of a healthy organization will be thwarted, leading to little effectiveness and poor performance. The ADOPT protocol and, more in general, MASs that provide accountability as a design property, would find immediate application in such contexts.

The paper is organized as follows. Section 2 explains the lack, due to relying mostly on goal-orientation, of current MAS approaches (in particular those concerning MAS organizations). It motivates the need of a change of perspective in order to enable the realization of accountability frameworks. Section 3 explains accountability, getting into the depths of computational accountability in organizational settings. Section 4 explains the ADOPT accountability protocol, including also the verification of properties that are granted by the protocol. Section 5 reports, as a case study, the implementation of ADOPT in the JaCaMo framework, which is one of the best-known, and widely used, MAS programming frameworks. Section 6 discusses the impact and related works, and Section 7 ends the paper with some conclusions.

2. Multiagent Systems Need Accountability

A common way to tackle coordination in MASs is to define an organization, that is, a "structure", within which interaction occurs. Inspired by the human organizations, a MAS organizational model includes a specification of organizational goals, and of a functional decomposition for achieving these goals through subgoals distributed among the agents in the organization. One key notion that is used in defining an organization is that of role. Organizational roles are typically understood as a way to delegate a complex task to different principals, abstracting from the actual individuals (the agents) who will play the roles. Following [25], when adopting a role, an agent, who satisfies some given requirements, acquires some powers inside that organizational context. A power extends the agent's capabilities, allowing it to operate inside the organization and to change the organizational state. Requirements, in turn, are needed capabilities that a candidate player must exhibit to play a particular role. This definition of role follows the one given in [26] in which roles are definitionally dependent

from the organization they belong to: Roles do not exist as independent entities, but rather, they are linked by definition to the organization they belong to.

Typically, in organization-oriented approaches, the MAS designer focuses on the organization of the system, taking into account its main objectives, structure and social norms. OperA [2], for instance, defines an organization along three axes: An organizational model describes the desired overall behavior of the organization, and specifies the global objectives and the means to achieve them; a social model defines the agent population, maps agents to organizational roles, and defines social contracts (agreements) on role enactment; and the interaction model correlates role-enacting agents through specified interactions. OMNI [3], based on OperA, provides a top-down, layered model of organization. An abstract level describes the general aims of the organization through a list of externally observable objectives. A concrete level specifies the means to achieve the objectives identified in the abstract level. Finally, an implementation level describes the activity of the system as realized by the individual agents. In this perspective, individual agents will enact organizational roles as a means to realize their own goals. E-institutions [1] model organizations as social groups of individuals built to achieve common (shared or antagonistic) goals. Here, organizations usually provide a set of norms, which agents in the system need to follow, depending on the role they are playing. If an agent violates a norm, the system must detect the violation and act consequently, for example, applying a sanction or warning the other participants. OCeAN [4] is a conceptual framework that extends the concept of the E-institution. The authors state that a suitable meta-model for open interaction systems must comprise, besides norms, an ontology, a set of roles associated to institutional actions, and a set of linguistic conventions. Roles provide agents the capability to perform institutional actions, which are actions whose meaning and effects require a convention among participants of the system. OCeAN foresees that institutional actions are messages, whose sending is bound to an institutional action by means of a count-as relation. 2OPL [10] is a rule-based language to define norms within a MAS, in order to achieve global goals of the system. The state of the world is represented by means of brute facts, first-order atoms, and of institutional facts which model the normative state of the system, tracing occurred violations. In Tropos [7], a MAS is seen as an organization of coordinated autonomous agents that interact in order to achieve common goals. Considering real-world organizations as a metaphor, the authors propose some architectural styles which borrow concepts from organizational theories, such as joint ventures. The styles are modeled using the notions of actor, goal and actor dependency, and are intended to capture concepts like needs/wants, delegations and obligations. Finally, the organizational model adopted in JaCaMo [11] decomposes the specification of an organization into three dimensions. The structural dimension specifies roles, groups and links between roles in the organization. The functional dimension is composed of one (or more) scheme(s) that elicits how the global organizational goal(s) is (are) decomposed into subgoals and how these subgoals are grouped in coherent sets, called missions, to be distributed to the agents. Finally, the normative dimension binds the two previous dimensions by specifying the roles' permissions and obligations for missions.

Recently, however, such a goal-oriented perspective on organizations has been criticized [13,27]. The point is that this approach provides a viable solution for modularizing complex tasks in terms of subtasks, each of which is assigned to a specific agent, but it is not a good solution for handling open distributed systems where agents are autonomous. In such scenarios, in fact, agents' goals might be in contrast to each other. Indeed, the organizational goals may not overlap completely with agents' goals. It follows that agents' obedience to the system norms cannot be taken for granted. Agent autonomy demands a different way of conceptualizing software modularity: not in terms of subgoals that are assigned to the agents, but rather in terms of responsibilities that are explicitly taken on by the agents. In [13], the authors see interaction as a central element of their modeling approach, and model interactions in terms of mutual expectations among the agents, bringing the conception of accountability as a way of characterizing the "good behavior" of each of the involved agents. Here, accountability is a directed relationship from one agent to another, and reflects the

legitimate expectations the second principal has of the first. The resulting approach, dubbed interaction-oriented software engineering (IOSE), focuses on social protocols, which specify how accountability relationships among the concerned principals progress through their interactions. In particular, as the authors themselves state, "in IOSE, it makes little sense to ask what functionality a role provides [...]; it makes sense though to ask to whom and for what is a role accountable [...]. A social protocol essentially describes how a principal playing a role would be embedded in the social world by way of accountability".

The essential message we want to stress, thus, is that the lack of an explicit treatment of the accountability relationship, within the existing organizational models, makes those models vulnerable. In this paper we have demonstrated this vulnerability in practice in the context of the JaCaMo platform. In particular, we show how the lack of an explicit take-on of responsibilities, for the distributed goals, thwarts goal achievement as well as the overall system capability of answering when some exceptional event occurs. We show, in fact, that when an agent joins an organization by playing a role, it is not aware of all the possible goals it will be asked to achieve as a role player. Thus, an agent can be assigned a goal for which it has no plan. The failure of the goal, though, cannot be attributed to the agent, nor to the organization, in an easy way. In fact, on the agent's side, the lack of a proper plan for the goal does not make the agent responsible. On the organization's side, the organization has no means to know what an agent can actually do, thus, the system cannot be considered as responsible either. In other terms, if an agent does not have the capabilities for achieving a goal, assigning that goal to it, even through an obligation, does not bring any closer the achievement of the goal of interest. In JaCaMo, the possibility to create goals along with the execution is a desired feature whose rationale is that goals pop up dynamically and cannot be foreseen; the problem is that agents, who are aware of their own capabilities, do not possess instruments for accepting or negotiating their goals.

Notably, these are not just features of JaCaMo, but of all the organizational models based on functional decomposition of goals. Those approaches substantially assume that the agents playing roles have been specifically designed for that purpose. This limits the openness of the system and the reuse of code. Instead, the design of an organization should rely on explicit relationships between agents, and also between agents and their organizations, capturing assumptions of responsibility by the agents. This would make the system accountable.

The organizational model, thus, should no longer be a structure that distributes goals to its agents, but it should become a way for coordinating responsibility assumption by the agents. More precisely, since agents are opaque (i.e., not inspectable even by the organization), and since no assumption on their capabilities can be done, an organization cannot assign a goal to an agent without putting the system in danger (as we have seen). The organization, however, can safely undertake an interaction protocol through which it negotiates with agents the attribution of goals. At the end of such a protocol, the agent itself takes on the responsibility of achieving a specific goal. The rationale here is that only the agent knows whether it has the control over a goal. Hence, if the agent accepts to bring about that goal, it also takes on the responsibility for the very same goal. The organization has therefore a legitimate expectation that the goal will be obtained. If the final outcome is not satisfactory, then, the agent is held to account for its conduct. Before explaining the accountability protocol, we characterize in the next section the key features of computational accountability.

3. Computational Accountability in Organizational Settings

Different research communities have dealt with the topic of accountability, such as [28–36]. While its main features remain relatively static, definitions vary in approach, scope and understanding in different communities. The cause of such variability lies with its socio–cultural nature and is one of the main reasons for the lack of a comprehensive support for the realization of accountability frameworks in current socio–technical systems. Accountability's most general definition refers to the assumption of responsibility for decisions and actions that a principal, individual or organization has towards another party. In other words, principals must account for their behavior to another

when put under examination. The concept is inherently social, and provides a mechanism by which entities constrain one another's behavior [17]. The previously cited examination is usually carried out by an investigative entity, a forum of auditors [37,38]. The process can be divided into three main phases: (1) The forum receives all information regarding the principals' actions, effects, permissions, obligations and so on that led to the situation under scrutiny; (2) the forum contextualizes actions to understand their adequacy and legitimacy; and finally, (3) the forum passes judgment on agents with sanctions or rewards. Our goal consists of automating the entire process for use in multiagent organizations, although we will presently leave out the sanctioning piece of the third phase due to its domain-specific application.

With the term *computational accountability*, we mean the abilities, realized via software, to trace, evaluate and communicate accountability, in order to support the interacting parties and to help solve disputes. In modern organizations, in particular, accountability determination can be a way to obtain feedback useful to evaluate and possibly improve the processes put in place and, eventually, the overall structure, too. Accountability determination is an extremely complex task, as the following example highlights.

Example 1. *Alice and Bob are a painter and a bricklayer, called by Carol for estimating the cost of renovating a room. The walls in the room are very old and for this reason, before being painted, they should be spackled. Since the prices seem reasonable, Carol decides to hire both Alice and Bob with the following work plan. Firstly, Bob should spackle the walls and Alice should paint them white afterwards. Come execution time, Bob decides to use a new variety of dark-colored spackle, since he has an open tin of it and he had not received any precise instruction from Carol. The following day, when Alice finds the dark colored walls she realizes she will not be able to satisfy the commitment she made with Carol because, in order to do a nice job, she will have to use twice as much paint as expected. This simple example shows many challenges brought about when trying to tackle accountability in a computational way. Let us suppose the agreement between Alice and Carol was somehow formalized (for example, with a contract). Alice is unable to fulfill her contract with Carol. Should the simple fact that she made a commitment, which is now impossible for her to fulfill, cause a computational system to conclude she is accountable for the failure? Clearly, conditions have changed since the original room inspection. One may argue that contextual conditions, constituting the prerequisites for Alice for the execution of the work, should have been formalized. In the real world, however, contextual conditions that hardly change over time are presumed implicitly stipulated even when they are not formalized. How could Alice foresee that Bob would have used a particular kind of spackle while the great majority of the bricklayers use white ones? Is, then, Carol accountable? Perhaps she should have checked all the involved parties to be in condition to fulfill their tasks properly when she organized the work. On the other hand, we know that Bob is the one who arbitrarily changed the color of the spackle. However, he received no instruction about the desired color of the spackle from Alice nor from Carol. In other words, there was no reason to assume that a similar decision would have caused problems. Here the problem arises from the fact that the involved parties, despite being in a collaborative environment, have different expectations and rely on (conflicting) assumptions about some contextual conditions. An alternative ending of the story is that Alice, feeling responsible, will paint the room at the agreed price because she values the satisfaction of the contract more than earning money.*

In our society, accountability becomes possible because of shared meanings culturally accepted as interpretations of events. Without such attributed meanings, mechanisms of accountability become difficult to be realized for lack of consensus concerning the interpretation of the events in question. This means that accountability becomes impossible in the absence of collective interpretations and meaning attributions. Another difficulty lies in value attribution. How does one decide whether a given outcome is "good" or "bad"? Who decides? This depends on the value attribution mechanism adopted in the system. In different systems the same outcome could be judged in different ways. In other words, accountability requires the presence of a social-meaning-defining structure in which actions and outcomes can be interpreted and evaluated in a uniform manner. Software systems can

provide such a structure in the form of an organization, which provides the tools and infrastructure needed to communicate and collaborate with the reassurance of shared meanings.

Our interest in applying accountability to the software world lies in two of its natural consequences: its diagnostic ability, that is, the ability to understand what went wrong, and its society-building aspect, that is, a system in which entities can know what most helps the encompassing organization and find encouragement to act on that knowledge to better it. With our concept of computational accountability, we strive towards a general goal of teaching agents correct behavior in the context of a particular organization. The mechanism of accountability contains two sides that we call a *positive* and a *negative* approach. Positive accountability means that an entity is socially expected to act in a certain way and will be held to account for that expectation's fulfillment. Negative accountability means that an entity is expected to not impede organizational progress and negatively impact others. In this work we focus on positive accountability, leaving the discussion related to negative accountability for future studies.

Accountability implies agency because if an agent does not possess the qualities to act "autonomously, interactively and adaptively", that is, with agency, there is no reason to speak of accountability, for the agent would be but a tool, and a tool cannot be held accountable [39]. As discussed in [40], entities are free to act as they wish even against behavioral expectations. Because accountability's domain lies in the past, it is only concerned with how entities acted, and places no restrictions on future actions. So, accountability does not hinder autonomy and allows entities the freedom to choose. On the other hand, neither does accountability unjustly single out those who have no control and are unable to act as autonomous beings. For example, during a robbery, a bank teller who hands over money would not be held accountable even though that person is an autonomous being who directly caused a financial loss, because that person had no "avoidance potential" [41], that is, no control over the situation. In an analogous fashion for positive accountability, entities must exhibit the possibility of action for accountable expectations: they must have control. Control is an extremely complex concept, related to the philosophical notion of free will. Restricting our attention to the scope of software agents, we cannot say, broadly speaking, that they have free will, but that they can exhibit some kind of control. Referring to [42], control can be defined as the the capability, possibly distributed among agents, of bringing about events. Ref. [43] gives a slightly different definition of control as the ability of an agent to maintain the truth value of a given state of affairs.

Accountability requires but is not limited to causal significance. Intuitively, for an entity to be held accountable for an outcome, that entity must be causally significant to that outcome [38]. However, accountability cannot be reduced only to a series of causes, that is, be reduced to the concept of traceability. Some authors, namely [40], even suggest that traceability is neither necessary nor sufficient for accountability. Accountability comes from social expectations that arise between principals in a given context. However, as implied by the word social, both principals must be aware of and approve the stipulated expectation in order to hold one another accountable. Accountability cannot be solely a result of principals' internal expectations. Expectations, here, follow the definition in [13], of ways to represent a given social state.

In order to make correct judgments, a forum must be able to observe the necessary relevant information. However, in order to maintain modularity, a forum should not observe beyond its scope. Organizations can be made up of other organizations. Societies, for example, contain micro-societies in which actions are encoded differently. In each micro-context the forum must be able to observe events and actions strictly contained in that context and decipher accountability accordingly. In each context, the forum must be able to observe events and actions strictly contained in its scope and decipher accountability accordingly. As context changes, accountability will change accordingly. Observability, thus, becomes integral for the forum to exercise its ability to process information. For this reason, a mechanism to compose different contexts and decide accountability comprehensively is essential.

The object under a forum's scrutiny can take the form of either an action or an outcome. The evaluation of the former implicates the recognition of social significance inherent in that action, relatively independently of where it leads. On the other hand, an evaluation of the latter denotes an importance in a state. When speaking of evaluating actions rather than outcomes in accountability, the examination implicitly involves a mapping between individual and social action, since the same individual action performed in different social contexts can take on different social significance. Therefore, a fundamental part of holding an individual accountable consists of identifying the social significance mapped from that individual's actions. In a social context an agent is accountable for an action to others, because of the realization that the others' goals depend on the outcome of the given actions.

When Does a MAS Support Accountability?

A MAS can be said to support accountability when it is built in such a way that accountability can be determined from any future institutional state. Consequently, the MAS must necessarily provide a structure that creates and collects contextualized information, so that accountability can actually be determined from any future institutional state. We consider integral to this process the following steps. A forum must receive all information (including all causal actions) regarding a given situation under scrutiny. The forum must be able to contextualize actions to understand their adequacy and legitimacy. Finally, the forum must be able to pass judgment on agents.

We identify the following necessary-but-not-sufficient principles a MAS must exhibit in order to support the determination of accountability.

Principle 1 All collaborations and communications subject to considerations of accountability among the agents occur within a single scope that we call organization.

Principle 2 An agent can enroll in an organization only by playing a role that is defined inside the organization.

Principle 3 An agent willing to play a role in an organization must be aware of all the powers associated with such a role before adopting it.

Principle 4 An agent is only accountable, towards the organization or another agent, for those goals it has explicitly accepted to bring about.

Principle 5 An agent must have the leeway for putting before the organization the provisions it needs for achieving the goal to which it is committing. The organization has the capability of reasoning about the requested provisions and can accept or reject them.

Principle 1 calls for situatedness. Accountability must operate in a specific context because individual actions take on their significance only in the presence of the larger whole. What constitutes a highly objectionable action in one context could instead be worthy of praise in another. Correspondingly, a forum can only operate in context, and an agent's actions must always be contextualized. The same role in different contexts can have radically diverse impacts on the organization and consequently on accountability attribution. When determining attribution, thus, an organization will only take into account interactions that took place inside its boundaries.

Placing an organizational limit on accountability determination serves multiple purposes. It isolates events and actors so that when searching for causes/effects, one need not consider all actions from the beginning of time nor actions from other organizations. Agents are reassured that only for actions within an organization will they potentially be held accountable. Actions, thanks to agent roles (Principle 2), also always happen in context.

To adequately tackle accountability by categorizing action, we must deal with two properties within a given organization: (1) An agent properly completes its tasks, and (2) an agent does not interfere with the tasks of others. The principles 2–5 deal more explicitly with the first property, that is, how to ensure that agents complete their tasks in a manner fair for both the agents and the organization. The second property is also partially satisfied by ensuring that, in the presence of goal dependencies, the first agent in sequence not to complete its goal will bear accountability, not only for its incomplete

goal, but for all dependent goals that will consequently remain incomplete. That is, should an agent be responsible for a goal on whose completion other agents wait, and should that agent not complete its goal, then it will be accountable for its incomplete goal and for that goal's dependents as well.

As an organizational and contextual aid to accountability, roles attribute social significance to an agent's actions. Following the tradition initiated by Hohfeld [44], a power is "one's affirmative 'control' over a given legal relation as against another." The relationship between powers and roles has long been studied in fields like social theory, artificial intelligence and law. By Principle 3 we stipulate that an agent can only be accountable for exercising the powers that are publicly given to it by the roles it plays. Such powers are, indeed, the means through which agents affect their organizational setting. An agent cannot be held accountable for unknown effects of its actions but, rather, only for consequences related to an agent's known place in sequences of goals. On the other hand, an agent cannot be held accountable for an unknown goal that the organization attaches to its role, and this leads us to Principle 4. An organization may not obligate agents to complete goals without prior agreement. In other words, an organization must always communicate to each agent the goals it would like the agent to pursue, and accountability will not be attributable in the presence of impossibilities, that is, when the agent does not have control of the condition or action to perform. Correspondingly, agents must be able to stipulate the conditions under which a given goal's achievement becomes possible, that is, the agent's requested provisions. The burden of discovery for impossibilities, therefore, rests upon an agent collective: A goal becomes effectively impossible for a group of agents should no agent stipulate a method of achievement. Conversely, an agent declares a goal possible the moment it provides provisions to that goal. Should a uniformed agent stipulate insufficient provisions for an impossible goal that is then accepted by an organization, that agent will be held accountable because by voicing its provisions, it declared an impossible goal possible. The opportunity to specify provisions, therefore, is fundamental in differentiating between impossibilities and possibilities.

The next section introduces a high-level protocol that enables the creation and collection of that contextualized information, which is necessary for accountability to be determined from any future institutional state. The adoption of this protocol allows an organization to support accountability.

4. The ADOPT Accountability Protocol

ADOPT is a protocol that allows the realization of accountable MAS organizations. Agents and organization will, thus, share the relevant information by exchanging messages, whose structure follows the FIPA ACL specification [45]. In the protocol, the organization is considered as a persona juris, a principal as any other principal, on which mutual expectations can be put. The protocol is divided into two main phases, a role adoption phase and a goal agreement one (explained in Sections 4.1 and 4.2, respectively), which are shown in Figures 1 and 2 as UML sequence diagrams. Such diagrams provide a sequencing of the messages, but what produces the accountability is the set of commitments that is created and that evolves with the messages, of both the role-adoption and the goal-agreement phases. Along with the message exchanges, in fact, the protocol records and tracks the evolution of accountability relationships between the parties, that we represent by way of social commitments [19,46]. In principle, other sequencings can be allowed as long as the the commitments are satisfied.

A social commitment is formally specified as $C(x, y, p, q)$, where x is the debtor, who commits to the creditor y to bring about the consequent condition q should the antecedent condition p hold. Social commitments embody the capacity of an agent to take responsibilities autonomously towards bringing about some conditions. They can be manipulated by the agents through the standard operations create, cancel, release, discharge, assign, delegate [19]. Commitment evolution follows the life-cycle formalized in [47]. A commitment is *Violated* either when its antecedent is true but its consequent is 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. A commitment should be *Active* when it is initially created. Active has two sub-states: *Conditional* as long as the antecedent does not occur,

and *Detached* when the antecedent has occurred. A commitment is autonomously taken by a debtor towards a creditor on its own initiative. This preserves the autonomy of the agents and is fundamental to harmonize deliberation with goal achievement. An agent will create engagements towards other agents while it is trying to achieve its goals or to the aim of achieving them. Commitments concern the observable behavior of the agents and have a normative value, meaning that debtors are expected to satisfy their engagements, otherwise a violation will occur. Commitment-based approaches assume that a (notional) social state is available and inspectable by all the involved agents. The social state traces which commitments currently exist and the states of these commitments according to the commitments life-cycle. By relying on the social state, an agent can deliberate to create further commitments, or to bring about a condition involved in some existing commitment.

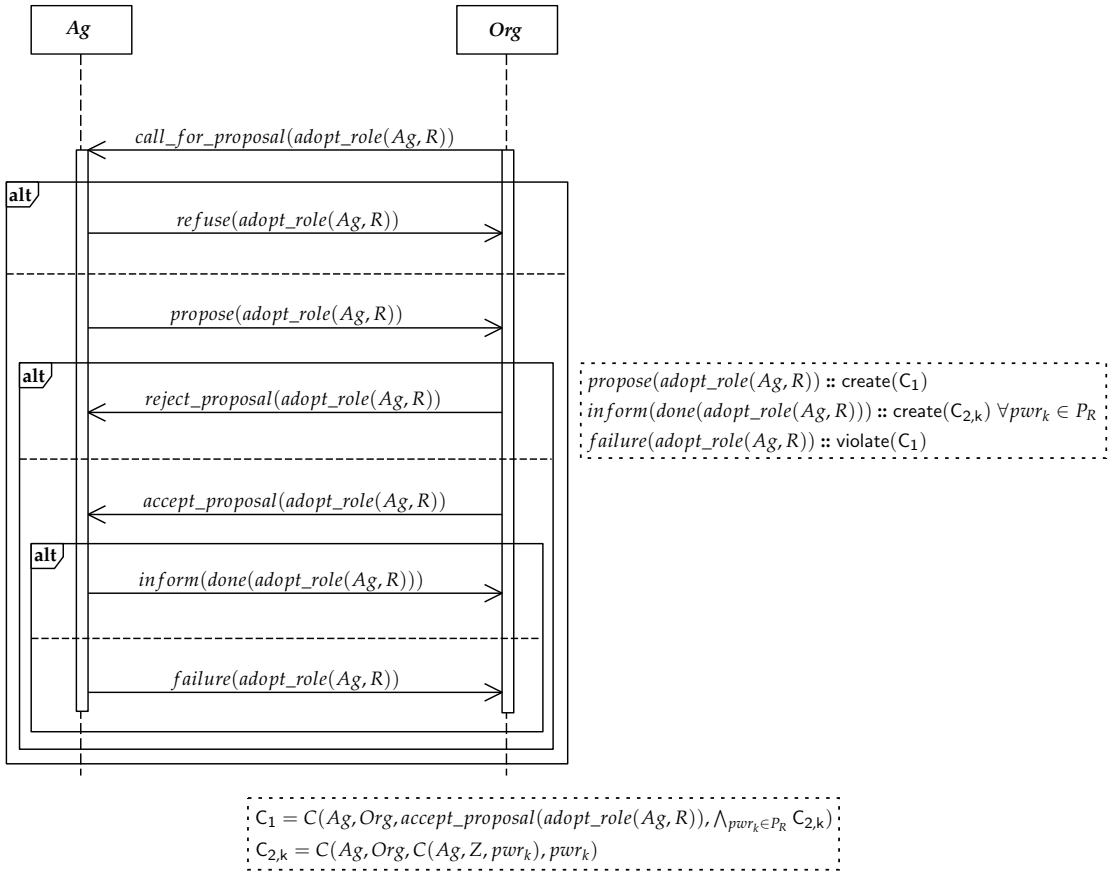

Figure 1. The first phase of the accountability protocol (role adoption).

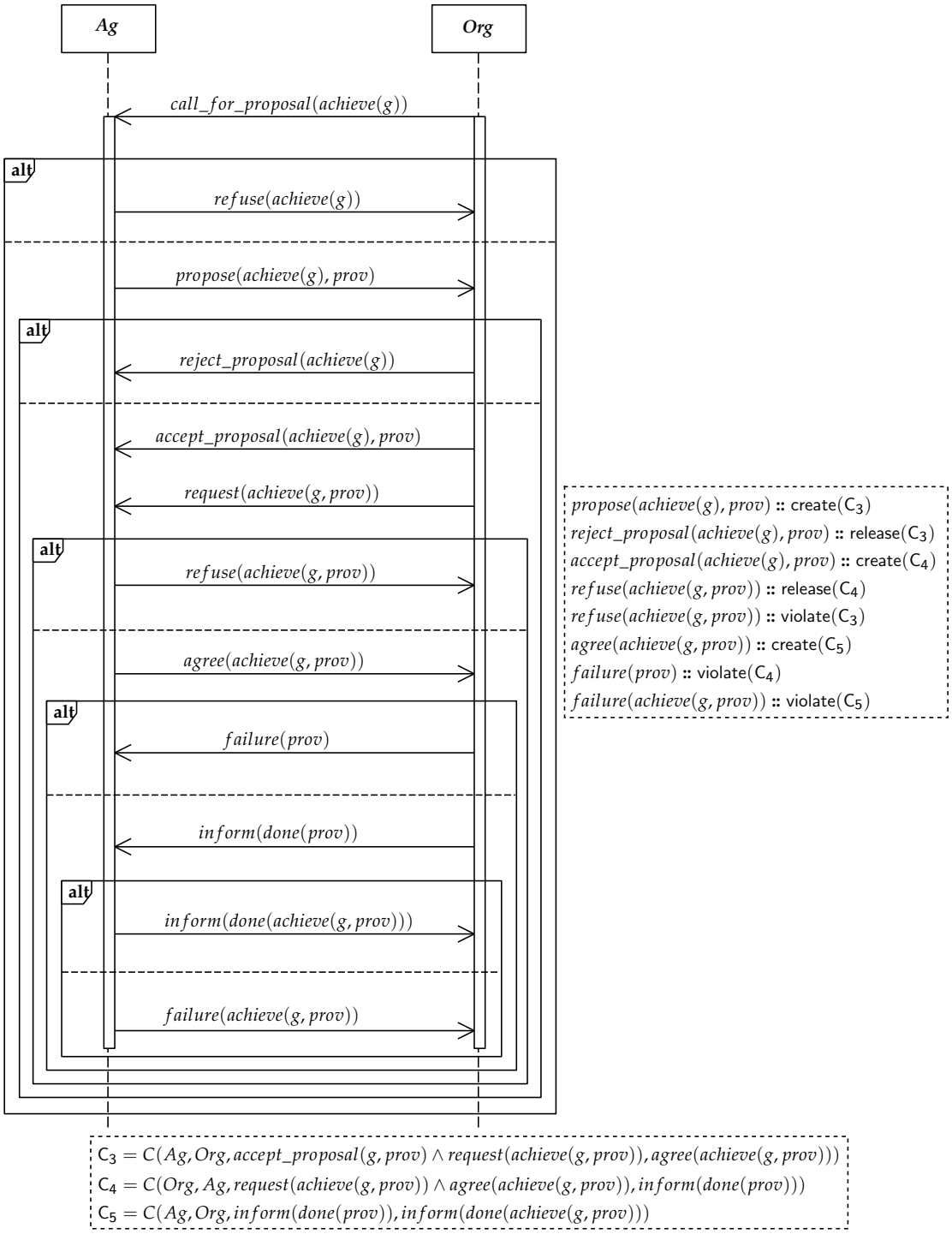

Figure 2. The second phase of the accountability protocol (goal agreement).

4.1. Role Adoption

The first phase of the protocol, reported in Figure 1, regulates the interaction that occurs when a new agent joins an organization by adopting an organizational role. The organization provides the context that gives significance to the actions that will be executed, thus satisfying Principle 1. Moreover, an agent will have an impact inside an organization only if the role adoption is successful, thus satisfying Principle 2. The interaction pattern follows the well-known Contract Net Protocol (CNP) [48,49]. (In the description of the messages we omit those arguments of the FIPA speech acts that

are not strictly necessary to manage accountability.) There are two different types of agents, one initiator and one or more participants. CNP provides a means for contracting as well as subcontracting tasks, where the initiator is the agent willing to delegate the task and participants are contractors. In our case, the organization is the initiator, the agents in the system are participants and the task corresponds to role adoption. The messages that are exchanged in this phase are the following:

- *call_for_proposal(adopt_role(Ag, R))*: this message notifies an agent Ag that the organization Org is looking for someone to play role R. A specification of R is provided, including the set P_R of the powers a player of that role will be provided with. Such powers allow a role player to operate in the organizational context, and the role player will have to commit towards the organization for their use, thus becoming accountable to the organization (see *propose*). We invoke a knowledge condition, and stipulate that an agent can only be accountable for exercising the powers that are publicly given to it by the roles it plays, thus realizing Principle 3.
- *refuse(adopt_role(Ag, R))*: with this message the agent declares that it is not interested in playing role R.
- *propose(adopt_role(Ag, R))*: this message is, instead, used by an agent to candidate for playing role R. This amounts to creating a commitment $\mathsf{C}_1 = C(Ag, Org, accept\_proposal(adopt\_role(Ag, R)), \bigwedge_{pwr_k \in P_R} C(Ag, Org, C(Ag, Z, pwr_k), pwr_k))$. In the following we denote by $\mathsf{C}_{2,k}$ the inner commitment concerning power pwr_k. Note that this commitment amounts to a declaration of awareness by the agent to the organization of the powers it has, and also that it will exercise such powers when requested by the legal relationships it will create towards other principals. In this way, the agent's behavior becomes accountable towards the organization itself.
- *reject_proposal(adopt_role(Ag, R))*: with this message the organization rejects the agent's proposal.
- *accept_proposal(adopt_role(Ag, R))*: with this message the organization accepts the agent as a player of role R. The commitment C_1, having the organization as creditor and the agent as debtor, will be detached.
- *inform(done(adopt_role(Ag, R)))*: this message from the agent notifies the organization that the agent has successfully adopted role R, and creates all the commitments $\mathsf{C}_{2,k}$ with $pwr_k \in P_R$. Commitment C_1 is satisfied.
- *failure(adopt_role(Ag, R))*: this message from the agent notifies the organization that the role adoption operation has not succeeded. This means that C_1 will not be satisfied.

We assume that *reject_proposal(adopt_role(Ag, R))* is mutually exclusive with *accept_proposal (adopt_role(Ag, R))*, and also that *failure(adopt_role(Ag, R))* is mutually exclusive with *inform (done(adopt_role(Ag, R)))*. This means that the occurrence of *reject_proposal(adopt_role(Ag, R))* will make C_1 expire. Instead, occurrence of *failure(adopt_role(Ag, R))* will cause the violation of C_1, neglecting what was previously stipulated.

After the role-adoption phase is successfully concluded, the organization can request the agents to pursue goals. This aspect is addressed in the second phase of the protocol.

4.2. Goal Agreement

This phase of the protocol regulates the agreement process between an agent and an organization for the achievement of a given organizational goal. Figure 2 shows the sequencing of the exchanged messages, which combines FIPA Request Interaction Protocol [50] with the already described CNP. When an organizational goal is to be pursued, the organization asks an agent to achieve it. The agent has, then, the possibility to accept or refuse the request made by the organization, or to make further requests of certain provisions that are considered by the agent as necessary to achieve the goal. This realizes Principles 4 and 5. The messages that are exchanged in this phase are the following:

- *call_for_proposal(achieve(g))*: by this message Org starts an interaction with Ag with the aim of assigning it a goal g to pursue.

- *refuse(achieve(g))*: the agent refuses the request made by the organization to achieve g.
- *propose(achieve(g), prov)*: with this message, an agent creates a commitment $\mathsf{C}_3 = C(Ag, Org, accept\_proposal(g, prov) \wedge request(achieve(g, prov)), agree(achieve(g, prov)))$. This means that on receiving a request to achieve g, given that the provisions *prov* were accepted, the agent is expected to agree to pursue the goal. It is up to the organization to decide whether *prov* is acceptable or not. Only in case *prov* is accepted, the commitment will eventually be detached. When no provisions are needed, *prov* will amount to $\top$.

This is a necessary step to reach two ends: agent's awareness of assigned goals, and agent's acknowledgement to exert the necessary control to reach the goal. The rationale is that an agent, by being aware of its own capabilities, will not promise to pursue a goal it is not capable to pursue—even indirectly by enticing other agents to act—and that the relevant conditions that are necessary for goal achievement, but that the agent does not control (*prov*), are clearly stipulated.

- *reject_proposal(achieve(g), prov)*: with this message the organization rejects the request for provisions *prov* made by the agent for goal (g). It will release C_3.
- *accept_proposal(achieve(g), prov)*: with this message the organization accepts to provide the provisions *prov* that were requested by the agent for goal g. This creates a commitment $\mathsf{C}_4 = C(Org, Ag, request(achieve(g, prov)) \wedge agree(achieve(g, prov)), inform(done(prov)))$. With this commitment the organization ties the request to the agent to pursue a goal with the supply of provisions *prov*. The commitment will be detached by the final word of the agent, and its possible agreement to the pursuit. The rationale is that, since provisions may come at a cost, before supplying them, the organization waits for a confirmation by the agent.
- *request(achieve(g, prov))*: with this message the organization *Org* asks agent *Ag* to achieve goal g, with provisions *prov*. Note that stipulation of what provisions should be supplied was done earlier through *propose* and *accept_proposal*, but while that was an interaction aimed at deciding whether to assign g to *Ag*, now *Org* wants the goal achieved. Generally, *request* may be uttered after *call_for_proposal*. The occurrence of a *request* contributes to detaching both C_3 and C_4.
- *refuse(achieve(g, prov))*: with this message the agent refuses to achieve g with provisions *prov*. It, thus, releases C_4, meaning that *Org* does not have to actually supply *prov*. We assume that this message is mutually exclusive with *agree(achieve(g, prov))*, thus, by this *refuse* the agent violates C_3, which at this point is already detached.
- *agree(achieve(g, prov))*: with this message the agent creates the commitment where $\mathsf{C}_5 = C(Ag, Org, inform(done(prov)), inform(done(achieve(g, prov))))$, meaning that it will pursue the goal if *prov* is actually provided. *Org* is now expected to supply provisions.
- *failure(prov)*: the organization did not succeed in supplying provisions. We assume this message to be mutually exclusive with *inform(done(prov))*, thus, C_5 will expire and C_4 will be violated.
- *inform(done(prov))*: the organization succeeded in supplying the provisions, thus, C_5 will be detached and C_4 will be satisfied.
- *inform(done(achieve(g, prov)))*: the agent achieved g and C_5 is satisfied.
- *failure(achieve(g, prov))*: the agent failed in achieving g with *prov*. Commitment C_5 is violated.

4.3. Verifying ADOPT

We now verify the correctness of the ADOPT protocol discussed above. To this end, we have to verify two aspects: first, the adherence of ADOPT to the five principles we have identified as necessary conditions for accountability; and second, the satisfaction of some fundamental properties that any good protocol should possess, specifically *safety* and *liveness* conditions. According to the literature about protocol verification [51–53], safety means that a protocol never enters an unacceptable state, whereas liveness, strictly related to state reachability, means that the protocol always progresses towards its completion. As usual in protocol verification, we will turn to a temporal logic for

formalizing and validating the protocol dynamics. In particular, the verification of ADOPT has to be focused on the treatment of the commitments that are created along the interaction, since these encode the relations upon which the accountability is established. Intuitively, we have to demonstrate that ADOPT allows the commitments to progress towards satisfaction. This demands for logics that are capable of handling commitments directly. El-Menshawy et al. [54] have proposed a temporal logic, named CTLC (i.e., computation-tree logics with commitments), which is an extension of CTL [55] with a modality operator for social commitments. In addition, the authors show how commitment-based protocols, specified in CTLC, can be checked by resorting to existing model-checking engines. Specifically, CTLC can be reduced to CTLK [56], an epistemic logic on branching time whose calculus has been implemented in the MCMAS model checker [57]. In a nutshell, the CTLC syntax is as follows:

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{EX}\varphi \mid \mathtt{EG}\varphi \mid \mathtt{E}(\varphi \mathtt{U} \varphi) \mid \mathtt{C}(i, j, \varphi),$$

where p is an atomic proposition, and where the temporal modalities have the same semantics as in CTL. For example, $\mathtt{EX}\varphi$ means that "there is a path where φ holds at the next state in the path", whereas $\mathtt{EG}\varphi$ means that "there is a path where φ holds in every state along that path". Finally, $\mathtt{E}(\varphi_1 \mathtt{U} \varphi_2)$ means that "there is a path along which φ_1 holds in every state until φ_2 holds". The same shortcuts defined for CTL are also valid in CTLC: $\neg\mathtt{EX}\neg\varphi \equiv \mathtt{AX}\varphi$ can be read as "φ holds in all the next states reachable from the current one"; $\neg\mathtt{EF}\neg\varphi \equiv \mathtt{AG}\varphi$: "$\varphi$ holds in every state of each path outgoing from the current state"; and $\neg\mathtt{EG}\neg\varphi \equiv \mathtt{AF}\varphi$: "$\varphi$ will eventually hold in every path outgoing from the current state".

The peculiar characteristic of CTLC is the modality $\mathtt{C}(i, j, \varphi)$, that is read: "agent i commits towards agent j to bring about proposition φ". Since the modality operator only tackles base commitments, a conditional commitment $\mathtt{C}(i, j, \psi, \varphi)$ must be reduced to the formula $\psi \rightarrow \mathtt{C}(i, j, \varphi)$, and in the following we will use $\mathtt{CC}(i, j, \psi, \varphi)$ as a shortcut of such a reduction. The interpretation of a CTLC formula is based on Kripke models where a specific accessibility relation R_{sc} is defined for the commitment modality $\mathtt{C}$. Providing a detailed reporting of the CTLC semantics is out of the scope of this paper. For our purposes, it is sufficient to say that a commitment modality $\mathtt{C}(i, j, \varphi)$ is satisfied in a model M at a state w iff φ is true in every accessible state from w using the accessibility relation R_{sc}.

For the sake of readability, we will just report some of the formulas that have actually been used for the verification of the ADOPT protocol. The usage of the CTLC language allows us to provide a synthetic yet formal account of how the protocol can be verified. Of course, in order to prove concretely the CTLC formulas above, we have to reduce them into equivalent CTLK formulas so as to use the MCMAS model checker. We leave these details out of this paper, as our intent, here, is to provide just some insights about the correctness of ADOPT, while the substantial contribution lies in the engineering of accountability in MAS enabled by ADOPT.

4.3.1. Principles 1, 2 and 3

Let us take into account the principles, and observe that the usage of commitments as a means for representing accountability relationships imposes some design choices. Commitments, in fact, are meaningful only when placed into a context, and the context is provided by the organization, the set of its roles, and its social state. Thus, when an agent is willing to play a role within an organization Org, it must be aware of the organization itself, the role R, and the powers P_R that come along with R. Only by having this knowledge, Ag can, during the role-adoption phase, create the commitment $\mathsf{C}_{2,k}$ for each of the powers pwr_k in P_R. Summing up: (1) The organization must exist; (2) roles must be defined in the context of an organization; and (3) power associated with roles must be known at the time of role enactment. When these elements are all known to an agent before joining an organization, the system implicitly satisfies the Principles 1, 2 and 3, which are structurally satisfied by the adoption of commitments as a means to represent accountability relations. Indeed, in the role-adoption phase of ADOPT, the only elements that come into play are the structural (i.e., static) properties of the organization.

4.3.2. Reachability Property and Principles 4 and 5

Principles 4 and 5, instead, concern goals, which are dynamic by nature. Specifically, Principle 4 states that an agent is only accountable for those goals for which it has taken an explicit commitment, whereas Principle 5 allows an agent to negotiate the provisions necessary for achieving a specific goal. The verification of these principles requires one to consider the dynamics of the accountability protocol, and overlaps with the *reachability* property we are interested in verifying in ADOPT. Relying on CTLC, it is possible to express useful properties about commitment satisfaction, and hence, about the correctness of the ADOPT protocol. For instance, for each commitment $C(i, j, \psi, \varphi)$ that is foreseen by the protocol, one can verify that when the commitment is created it can also be satisfied. This corresponds to expression of the following reachability property in CTLC: $\mathtt{AF}\ \mathtt{EF}(\mathtt{createdC} \rightarrow \mathtt{CC}(i, j, \psi, \varphi))$, where `createdC` is an atomic proposition that becomes true when the given commitment $C(i, j, \psi, \varphi)$ is created (i.e., when a specific message is sent by the debtor agent). This formula is valid if and only if in all the possible paths (i.e., runs of ADOPT), a state will be reached from which there exists at least one path along which either `createdC` is false (the commitment is not created), or `createdC` is true and the modality $\mathtt{CC}(i, j, \psi, \varphi)$ will eventually be satisfied. Please recall that $\mathtt{CC}(i, j, \psi, \varphi)$ is just a shortcut for $\psi \rightarrow \mathtt{C}(i, j, \varphi)$. This formula is satisfied either when ψ is false, corresponding to the case in which the commitment expires (for example, released by the creditor); or when ψ is true and hence $\mathtt{C}(i, j, \varphi)$ must hold. The latter corresponds to the case in which the commitment is, firstly, detached by the creditor and, then, satisfied by the debtor. Intuitively, this can be verified simply by looking at the sequence diagrams in Figures 1 and 2, and observing that whenever a commitment is created by a message of the debtor, the creditor has always the chance to release that commitment or detach it. Similarly, whenever a commitment is detached, the protocol encompasses at least a run along which the debtor can send a message whose meaning consists of the progression of that commitment to satisfaction.

In short, it is possible to show that the above formula holds for every commitment that may arise in ADOPT. This means that in every possible run of ADOPT, there is always an execution path along which the conditional commitment $C(i, j, \psi, \varphi)$, once created, can always be released by the creditor, or satisfied by the debtor. As a consequence, we can conclude that Principles 4 and 5 of the accountability requirements are actually satisfied, since the agent will have also the chance to agree to achieve a goal by creating a commitment (specifically C_4 of the goal-agreement phase), and by negotiating its provisions again via a commitment (specifically C_3). An agent, thus, will only be accountable for the goals it has agreed upon, possibly having also established some necessary provisions.

4.3.3. Safety

To verify the safety condition, we have to show that "bad" conditions will never be reached. In the specific case of ADOPT, an unwanted condition is the impossibility to satisfy a commitment $C(i, j, \psi, \varphi)$ after the commitment has been created. This condition can be formulated in CTLC as the formula $\neg\mathtt{AF}\ \mathtt{EF}(\mathtt{createdC} \rightarrow \mathtt{CC}(i, j, \psi, \varphi))$. Also in this case, we can show intuitively that the above formula is false in every possible run of the protocol. In fact, the only situation in which the role-adoption phase completes without the satisfaction of any commitment is when the agent answers with a *refuse* message to the call for proposal from the organization, but in this case no commitment is created. In all the other possible runs of the first phase, for every commitment that is created there exists at least one run along which it will be eventually satisfied. A similar consideration holds for the goal-agreement phase, too. This means that ADOPT is safe.

4.3.4. Liveness and Nested Commitments

The nested commitments in ADOPT also deserve some special attention. In the role-adoption phase, the following nested commitments are created: $\mathsf{C}_1 = C(Ag, Org, accept\_proposal(adopt\_role$

$(Ag, R))$, $\bigwedge_{pwr_k \in P_R} \mathsf{C}_{2,k}$); where, as before, $\mathsf{C}_{2,k} = C(Ag, Org, C(Ag, Z, pwr_k), pwr_k)$ for every power $pwr_k \in P_R$. In this case it is interesting to verify whether whenever C_1 is detached by Org, Ag has the chance to satisfy C_1 by creating the commitments $\mathsf{C}_{2,k}$. That is, whether there exists at least one run of the protocol where these commitments will be actually created. This corresponds to verifying the *liveness* of the protocol by asking whether something "good" will eventually happen. The following CTLC formula serves this purpose: $\mathtt{AG}(\mathtt{detachedC1} \rightarrow \mathtt{EF}(\mathtt{createdC2k}))$, where `detachedC1` is an atomic proposition that becomes true when Org sends message *propose(adopt_role(Ag,R))*, whereas `createdC2k` is another proposition that becomes true when Ag sends the message *inform(done(adopt_role(Ag, R)))*. Indeed, looking at the sequence diagram in Figure 1, it is apparent that once the commitment C_1 is detached, there exists at least one run of the protocol along which the message *inform(done(adopt_role(Ag, R)))* is actually sent by the agent. The formula is thus satisfied, and the protocol enjoys the liveness property.

5. Case Study: Accountability in the JaCaMo Framework

JaCaMo [11] is a conceptual model and programming platform that integrates agents, environments and organizations. It is built on the top of three platforms, namely Jason [8] for programming agents, CArtAgO [9] for programming environments, and Moise [58] for programming organizations. More specifically, Jason is a platform for agent development based on the language AgentSpeak [59]. Here, an agent is specified by a set of beliefs, representing both the agent's current state and knowledge about the environment, a set of goals, and a set of plans which are courses of actions, triggered by events. CArtAgO, based on the Agents & Artifacts meta-model [60], is a framework for environment programming which conceives the environment as a layer encapsulating functionalities and services that agents can explore and use at runtime [61]. An environment is programmed as a dynamic set of artifacts, whose observable states can be perceived by the agents. Agents can act upon artifacts by executing the operations that are provided by the artifact's usage interface. Finally, Moise implements a programming model for the organizational dimension. It includes an organization modeling language, an organization management infrastructure [62] and a support for organization-based reasoning at the agent level. A JaCaMo multiagent system is, then, given by an agent organization, programmed in Moise, organizing autonomous agents, programmed in Jason, working in a shared, artifact-based environment, programmed in CArtAgO.

According to [62], the Moise organizational model, adopted in JaCaMo, decomposes the specification of an organization into three dimensions. The structural dimension specifies roles, groups and links between roles in the organization. The functional dimension is composed of one or more schemes that elicit how the global organizational goals are decomposed into subgoals and how these subgoals are grouped in coherent sets, called missions, to be distributed to the agents. Finally, the normative dimension binds the two previous dimensions by specifying the roles' permissions and obligations for missions. One important feature of Moise is to avoid a direct link between roles and goals. Roles are linked to missions by means of permissions and obligations, thereby keeping the structural and functional specifications independent. This independence, however, is source of problems from the viewpoint of accountability determination. The reason is that schemes, in principle, can be dynamically created during the execution (thus modifying the organizational specification), and assigned to groups within an organization when agents are already playing the associated roles. This means that agents, when entering into an organization by adopting an organizational role, in principle have no information about what they could be asked to do in the future. At the same time, the organizational infrastructure implemented in JaCaMo does not provide any mechanism for agents to explicitly accept a given organizational goal or to negotiate any provision for it. This is, indeed, in contrast with what was discussed in Section 3.

JaCaMo provides various kinds of organizational artifacts that altogether allow encoding the state and behavior of an organization, in terms of groups, schemes and normative states. These organizational artifacts provide both the actions the agents will use (to take part in

an organization and act upon it) and the observable properties that allow the state of the organization (and its evolution) to be perceived by the agents. Artifacts' observable properties are automatically mapped to agents' beliefs. The main organizational artifacts are the following:

- **OrgBoard** artifacts, which keep track of the overall current state of the organizational entity, one instance for each organization (optional);
- **GroupBoard** artifacts, which manage the life-cycle of specific groups of agents, one for each group;
- **SchemeBoard** artifacts, each of which manages the execution of one scheme;
- **NormativeBoard** artifacts, used to maintain information concerning the agents' compliance to norms.

5.1. Accountability Issues and Their Reasons

In order to explain the lack of accountability of JaCaMo, let us now consider a scenario based on an excerpt of the *building-a-house* example presented in [11]:

Example 2. *An agent, called Giacomo, wants to build a house on a plot. In order to achieve the goal he will have to hire some specialized companies, and then ensure that the contractors coordinate and execute in the right order the various subgoals. Some tasks depend on other tasks while some other tasks can be performed in parallel. This temporal ordering is specified in the functional specification of the scheme describing the process.*

As soon as the building phase starts, Giacomo creates a GroupBoard artifact, called here `bh_group` (for sake of discussion, the example is slightly revised with respect to the original version, presented in [11]. This includes a change in the artifacts' names), following the organization specification. Roles are gathered in a group that will, then, be responsible for the house construction. After that, he adopts his role and asks the hired agents to adopt theirs. Roles are adopted by executing an operation that is provided by the GroupBoard artifact. Finally, a SchemeBoard artifact, called `bh_scheme`, is created. When all agents have adopted their roles (i.e., when the group is well-formed), this is added to the schemes the group is responsible for.

After this step, the involved agents could be asked to commit to some "missions"—in JaCaMo, the term "commit" does not refer to social commitments but it is used in a general sense. This is done by relying on obligations that are issued by the organization, according to the normative specification. Figure 3 shows the general interaction pattern which is used in JaCaMo for role adoption, mission distribution and goal assignment, as it would be instantiated for a `companyA` agent, a `plumber` role and an `install_plumbing` mission (which in turn contains a `plumbing_installed` goal). Agent `companyA` is asked to commit to `install_plumbing` with an obligation of the form `obligation(companyA, n8, committed(companyA, install_plumbing, ...), ...)`, where `n8` is the norm that binds the `plumber` role with the `install_plumbing` mission in the normative specification. This does not yet mean the agent has to pursue the goal; this other obligation, however, may now be issued by the organization. Indeed, the main purpose of the SchemeBoard artifact, `bh_scheme`, is to keep track of which goals are ready to be pursued and create obligations for the agents accordingly. For instance, when the `plumbing_installed` goal will be ready to be pursued, `companyA` will receive a new an obligation `obligation(companyA, ..., achieved(..., plumbing_installed, companyA),...)`, and so forth with other organizational goals as soon as they become ready. Such obligations are observed by the agents and the corresponding internal goals are automatically created.

Listing 1 shows an excerpt of the `companyA` agent. The above-mentioned obligation, through the plan at line 15, creates the (internal) goal that is then pursued by following the plan at line 21. After that, the organizational goal is set as achieved, too (line 19), using the `goalAchieved` operation provided by the SchemeBoard artifact.

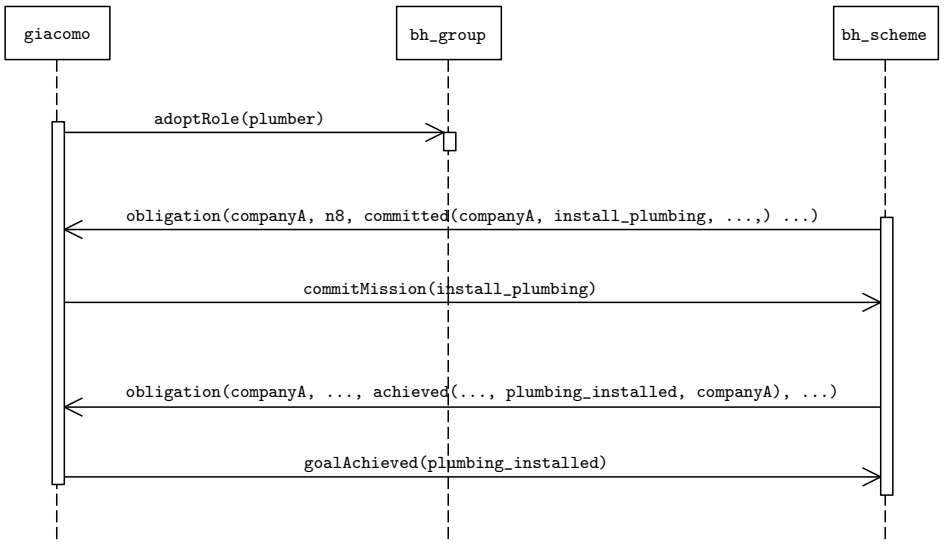

Figure 3. Interaction between the `companyA` agent and the organization in the *building-a-house* example.

Listing 1. Excerpt of the Jason code of the `companyA` agent.

```
1  task_roles("Plumbing", [plumber]).
2
3  +!contract(Task,GroupName)
4     : task_roles(Task,Roles)
5    <- lookupArtifact(GroupName, GroupId);
6       for (.member(Role, Roles)) {
7         adoptRole(Role)[artifact_id(GroupId)];
8         focus(GroupId)
9       }.
10
11 +obligation(Ag,Norm,committed(Ag,Mission,Scheme),Deadline)
12    : .my_name(Ag)
13   <- commitMission(Mission)[artifact_name(Scheme)].
14
15 +obligation(Ag,Norm,What,Deadline)[artifact_id(ArtId)]
16    : .my_name(Ag) &
17      (satisfied(Scheme,Goal)=What | done(Scheme,Goal,Ag)=What)
18   <- !Goal[scheme(Scheme)];
19      goalAchieved(Goal)[artifact_id(ArtId)].
20
21 +!plumbing_installed
22   <- installPlumbing.
```

Now, let us suppose that Giacomo decides to have an air conditioning system installed, a thing he had initially not thought of for this house. Suppose also that he wants to exploit the contracted companies to achieve this purpose, which is related to the house construction, but was not discussed. Let us also suppose he decides to assign an `air_conditioning_installed` goal to the agent playing the `plumber` role. Giacomo's exploitive plan would work because when an agent adopts a role in a group, that agent has no information about the tasks that could be assigned to it. The `bh_scheme` SchemeBoard artifact could even not have been created yet. In fact, tasks could be created independently of roles, and only subsequently associated with them. In the example, however, the `companyA` agent, playing the `plumber` role reasonably will not have a plan to achieve the `air_conditioning_installed` goal (indeed, it has no such plan). Thus, when the corresponding obligation is created, this will not be fulfilled.

Given the above scenario, who could we consider accountable for the failure of the organizational goal `air_conditioning_installed`?

- Should the agent playing the `plumber` role be held accountable? The agent violated its obligation but it could not have reasonably anticipated the goal's introduction, which effectively made achievement impossible.
- Should Giacomo be held accountable since he introduced an unachievable goal, however licit?
- Perhaps the system itself ought to bear the brunt of accountability since it permits such kind of behavior? The system, however, does not know agent capabilities.

The inability to attribute accountability stems from the lack of adherence to Principles 4 and 5. Goal assignment is, in fact, performed through schemes, which can even be dynamically created and associated with an existing group. Moreover, the very independence between roles and goals violates Principle 4: When enacting a role in JaCaMo, agents do not have a say on the kind of goals they could be assigned. For this reason they cannot be held accountable later for some organizational goal they have not achieved. The problem, here, is that JaCaMo's organizational infrastructure agents do not have the possibility to discuss with the organization about the acceptability of organizational goals, which is, instead, encoded in the goal-agreement phase of the ADOPT protocol. In particular, it is impossible for agents to put before the organization the provisions needed to achieve a given organizational goal. This contradicts Principle 5.

Another critical issue concerns the powers the agents gain by taking part in an organization. Following JaCaMo's conceptual meta-model, the only ways for an agent to affect the organizational state are either to enter into a group, or to change the state of an organizational goal, for example, by achieving it. In order to join a group, an agent must have access to the GroupBoard artifact which manages that group—indeed, the `adoptRole` operation is provided by that artifact. Similarly, to set an organizational goal as achieved, the agent must have access to the SchemeBoard artifact, which manages the execution of the scheme to which the goal belongs. From an accountability standpoint, then, the role adoption does not end when a given agent enters into a group, but only when a scheme is associated with the group and, consequently, the agent gains access to it. However, should a new scheme be added to the group, the agents inside the group would also receive new powers (i.e., the powers to change the states of the goals in the new scheme). This clearly contradicts Principle 2, which requires a declaration of awareness of the powers, associated with a role, that should be done by each of its players. This awareness is, instead, guaranteed by the role-adoption phase of the ADOPT accountability protocol.

5.2. Achieving Accountability

We now show how we implemented the ADOPT accountability protocol in JaCaMo. A new kind of artifact, called *AccountabilityBoard* artifact, was added to the organizational infrastructure to preserve modularity. (The source code of the AccountabilityBoard artifact can be found here: http://di.unito.it/accountabilityboard.) In other words, this addition did not affect the implementation of the other organizational artifacts that were previously in JaCaMo. The new artifact supports both phases of ADOPT: the role-adoption phase and the goal-agreement phase. The artifact also provides some observable properties, informing the agents that focus on it about the overall state of the interaction. These observable properties represent the events occurring during the interaction, and allow encoding of the created commitments. More precisely, the artifact keeps track, in the first phase, of which calls for roles are pending, of the agents' replies, of which proposals have been accepted (and which have not), of which agents have successfully completed the role adoption, and which have failed. Later on, in the goal-agreement phase, the observable properties encode which provisions were accepted and which were not, which goals were agreed upon by which agents (and with what kinds of provisions), which ones were refused, and which provisions were confirmed to be holding. All such observable properties are created by executing the above described operations on the accountability artifact. In case of inspection, the observable properties help to identify the accountable parties.

Concerning the role-adoption phase, the artifact provides an organization with the operations to call for role players, and to accept or reject the agents' proposals. Agents, on the other hand,

are provided with the operations to answer calls (either by refusing, or by proposing themselves), to declare the acquisition of organizational powers and to declare a failure in the role-adoption process. With respect to the power awareness declaration, we assume here that an agent acquires the powers to operate on a given scheme when the scheme is assigned to the group to which the agent belongs, and when the agent commits to the missions associated with its role in the scheme itself. To stipulate this awareness, the agent declares it has access both to the group and to the scheme artifacts, as player of the given role, which together define the scope of the agents' actions in the organizational context. By doing so, the agent ensures that it is conscious that it could be requested to achieve organizational goals defined in the given scheme as a role player in the given group (and it will have the possibility to refuse them or agree to their achievement). Should a new scheme be added to a given group, agents inside it should renegotiate the terms of role adoption (by executing again the first phase of the protocol) in order to acquire the organizational powers needed to act on the new scheme, too. The *AccountabilityBoard* artifact operations that allow all these things are:

- `callForRole(String addressee, String role)`: implements ADOPT's *call_for_proposal(adopt_role(Ag, R))*. It allows the organization's owner to open a call for a role, addressed to a given agent. An observable property `pendingRole(addressee, role)` is defined;
- `refuseRole(String role)`: implements *propose(adopt_role(Ag,R))*. It allows agents to answer to a call for role, refusing to adopt it. Its execution creates an observable property `refusedRole(agent, role)`;
- `proposeForRole(String role)`: implements *propose(adopt_role(Ag,R))*, and allows agents to propose themselves as role players. It creates an observable property `proposedForRole(agent, role)`;
- `rejectProposalForRole(String proposer, String role)`: implements *propose(adopt_role(Ag, R))*, and allows an organization's owner to reject a `proposer` agent's proposal. An observable property `roleProposalRejected(proposer, role)` is created;
- `acceptProposalForRole(String proposer, String role)`: implements *accept_proposal(adopt_role(Ag, R))*, and allows the organization's owner to accept an agent's proposal. It creates an observable property `roleProposalAccepted(proposer, role)`;
- `declareAdoptionSuccess(String role, String group, String scheme)`: implements *inform(done (adopt_role(Ag, R)))*, and allows a role player to declare awareness of the powers given by scheme `scheme`, belonging to group `group` as a player of role `R`. An observable property `powerAcquired(agent, role, group, scheme)` is defined. The execution of this operation completes the role-adoption phase of ADOPT;
- `declareAdoptionFailure(String role, String group, String scheme)`: implements *failure(done (adopt_role(Ag, R)))*, and allows an agent to inform the organization that its role-adoption process failed. An observable property `powerAcquisitionFailed(agent, role, group, scheme)` is defined.

With respect to the goal-agreement phase of ADOPT, instead, the artifact provides operations to the agents for refusing a goal, for proposing themselves as goal achievers, for proposing provisions for it, for agreeing to pursue a goal when the provisions are accepted by the organization, and for declaring goal achievement or failure. At the same time, it allows the organization (i.e., the agent that is the organization's owner) to open a call for goal achievement, to accept or reject provisions proposed by the agents, and to declare that a given provision holds or not. In particular, the operations provided by the artifact are:

- `callForGoal(String addressee, String goal)`: implements *call_for_proposal(achieve(g))* and allows the organization's owner to open a call for goal `goal` addressed to agent `addressee`, a `pendingGoal(addressee, goal)` observable property is defined;
- `refuseGoal(String goal)`: implements ADOPT's *refuse(achieve(g))* and *refuse(achieve(g,prov))*. With this operation, an agent that a goal is addressed to can refuse to achieve it. A `refusedGoal(agent, goal)` observable property is defined;

- `proposeProvision(String goal, String provision)`: realizes the *propose(achieve(g), prov)* message. An agent can propose a provision `provision` for goal `goal` assigned to it. A `pendingProvision (agent, goal, provision)` observable property is created;
- `acceptProvision(String proposer, String goal, String provision)`: implements *accept_proposal (achieve(g), prov)* and allows the organization's owner to accept a provision `provision` proposed by agent `proposer` for goal `goal`. An observable property `acceptedProvision(proposer, goal, provision)` is defined;
- `rejectProvision(String proposer, String goal, String provision)`: conversely, implements *reject_proposal(achieve(g), prov)*, and allows the organization's owner to reject the provision proposed by the agent for the goal. A `rejectedProvision(proposer, goal, provision)` property is defined;
- `requestGoal(String addressee, String goal)`: implements *request(achieve(g,prov))* in ADOPT. The organization's owner asks `addressee` to agree to achieve `goal` since the provisions proposed by it have been accepted. A `requestedGoal(addressee, goal)` property is created;
- `agreeGoal(String goal)`: implements *agree(achieve(g, prov))* and allows an agent to agree to achieve a given goal previously requested. An `agreedGoal(agent, goal)` property is defined;
- `informProvision(String goal, String provision)`: implements *inform(done(prov))*. The organization's owner agent confirms that a provision `provision` related to a goal `goal` is currently holding. This operation defines an observable property `holdingProvision (goal, provision)`;
- `failureProvision(String provision)`: implements *failure(prov)*. The organization's owner agent declares that a provision `provision` related to a goal `goal` cannot hold. This operation defines an observable property `failedProvision(provision)`.
- `informGoal(String goal)`: implements *inform(done(achieve(g,prov)))*. It allows an agent to confirm that a goal has been achieved. This operation defines an observable property `achievedGoal(goal)`;
- `failureGoal(String goal)`: implements *failure(achieve(g,prov))*. The agent declares that it has not been able to achieve goal `goal`. This operation defines an observable property `failedGoal(goal)`.

5.3. Building-a-House Revisited

Let us, now, see how the AccountabilityBoard artifact supports the execution of the ADOPT protocol with the help of a revised version of our example. (The full code of the revised example is available here: http://di.unito.it/buildingahouse.) In this case, before starting with the actual house construction, Giacomo will create an instance of the AccountabilityBoard artifact, thereby becoming the organization's owner. This will allow it to execute organization-reserved operations on the artifact, such as `acceptProposal`, `acceptProvision`, `confirmProvision`, and so on.

The original implementation of the example presented in [11] also includes a contracting phase before the building one, in which Giacomo hires the needed company agents and assigns roles to them through an auction mechanism. Thus, we mainly focus here on the agreement of goals. With respect to the first part of the protocol, however, agents, after having adopted their roles in the `bh_group`, still have to declare their awareness of the powers they are endowed with when the `bh_scheme` is assigned to the group. This is achieved by means of the plan reported in Listing 2. As soon as the scheme is added to the ones the `bh_group` is responsible for, agents inside it will focus on it (lines 7 and 11) and declare power awareness with respect to the `bh_group` and `bh_scheme` (line 8). At the same time, they will commit to their missions as requested by the organizational infrastructure.

Role players will now have the possibility to explicitly refuse or agree to pursue the organizational goals assigned to them (possibly asking for some provisions to hold) when asked by the organization's owner. This is achieved by means of the plans reported in Listing 3. For instance, the plan at line 2 is triggered when an acceptable goal (line 4) is proposed to the agent and there is no need to ask for provisions (line 5). In this case, the agent will simply propose a dummy *true_prov* provision to achieve the goal (line 6). The plan at line 9 deals with an unacceptable goal (line 11). In this case the agent

simply refuses it (line 12). The plan at line 15 allows the agent to propose a provision for a pending goal (line 17). Should the provision be accepted (line 22) and the goal requested (line 20), the agent would then agree to pursue the goal given the provision (line 23). Finally, the plan at line 26 is triggered when a provision for a previously agreed-upon goal is declared to hold. In this case the agent works in order to achieve the goal. An internal goal corresponding to the organizational one will be generated 30 and, if achieved, the organizational goal will be set to achieved (line 31), as well. As a final step, the goal must be declared as achieved on the AccountabilityBoard, too (line 32).

Listing 2. Jason plan needed by an agent to declare powers awareness.

```
1  //declare powers awareness
2  +schemes(L)[artifact_name(_,Group), workspace(_,_,W)]
3     : .my_name(Ag) &
4       play(Ag,Role,Group)
5    <- for (.member(Scheme,L)) {
6         lookupArtifact(Scheme,ArtId)[wid(W)];
7         focus(ArtId)[wid(W)];
8         declareAdoptionSuccess(Role,Group,Scheme);
9         .concat(Group,".",Scheme,NBName);
10        lookupArtifact(NBName,NBId)[wid(W)];
11        focus(NBId)[wid(W)];
12      }.
```

Listing 3. Excerpt of the Jason plans needed by an agent playing a role in an organization to be compliant with the goal-agreement phase of ADOPT.

```
1  //no provision needed
2  +pendingGoal(Ag,Goal)
3     : .my_name(Ag) &
4       acceptableGoal(Goal) &
5       not provision(Goal,Prov)
6    <- proposeProvision(Goal, true_prov).
7
8  // refuse goal
9  +pendingGoal(Ag,Goal)
10    : .my_name(Ag) &
11      not acceptableGoal(Goal)
12   <- refuseGoal(Goal).
13
14 //propose provision
15 +pendingGoal(Ag,Goal)
16    : .my_name(Ag) & acceptableGoal(Goal) & provision(Goal,Prov)
17   <- proposeProvision(Goal,Prov).
18
19 //agree goal, provision accepted
20 +requestedGoal(Ag,Goal)
21    : .my_name(Ag) &
22      acceptedProvision(Ag,Goal,_)
23   <- agreeGoal(Goal).
24
25 //provision confirmed, must satisfy goal
26 +holdingProvision(Goal,Prov)
27    : .my_name(Ag) &
28      agreedGoal(Ag,Goal) &
29      acceptedProvision(Ag,Goal,Prov)
30   <- !Goal[scheme(Scheme)];
31      goalAchieved(Goal)[artifact_id(ArtId)];
32      informGoal(Goal).
```

Listing 4, in turn, shows an excerpt of the plans needed by Giacomo, as organization owner, to be compliant with the second phase of the ADOPT protocol. First of all, the plan at line 1 allows it to open a call for the achievement of a given goal (line 3) when the corresponding obligation is issued by the organizational infrastructure. The plans at lines 5 and 10, in particular, allow it to accept or

reject provisions for goals asked by the role players (lines 7 and 12) depending on whether they seem acceptable or not (lines 6 and 11). If a provision is accepted, then the goal is requested to the agent, too (8). The plans at lines 14 and 19, finally, allow the organization's owners to inform that a given provision which has been agreed by some agent for a goal is holding. To this end, the provision must be present in the agent's belief base (i.e., the agent must believe that the proposition representing the provision is true).

Going back to the *companyA* agent, playing the *plumber* role, should an *air_conditioning_installed* (*aci* for short) goal be requested via *call_for_proposal(achieve(aci))*, the agent would have the legitimate possibility to refuse the proposal or to conditionally accept by specifying a provision. It could therefore ask that the walls be built in order to successfully install the air conditioning by adding *provision(aci, walls_built)*. The plan in line 15 in Listing 3 would be consequently triggered. As a result of executing `proposeProvision`, the following commitment would be created: $C_1 = C(companyA, Giacomo, accept\_proposal(aci, walls\_built) \wedge request(achieve(aci, walls\_built)), agree(achieve(aci, walls\_built)))$. Giacomo in turn has the freedom to either accept or reject *companyA*'s proposal. Since in the scheme specification walls are built beforehand, Giacomo will reasonably accept the provision by executing the `acceptProvision` operation in line 7 of Listing 4, leading to the creation of the commitment $C_2 = C(Giacomo, companyA, request(achieve(aci, walls\_built)) \wedge agree(achieve(aci, walls\_built)), inform(done(walls\_built)))$. Moreover, it will subsequently execute `requestGoal`, which in turn detaches C_1 and partially satisfies the antecedent in C_2.

Listing 4. Jason plans needed by an organization's owner agent to be compliant with the second phase of the accountability protocol.

```
1  +obligation(Ag,_,What,_)[artifact_id(ArtId)]
2     : (satisfied(Scheme,Goal)=What | done(Scheme,Goal,Ag)=What)
3    <- callForGoal(Ag,Goal).
4
5  +pendingProvision(Agent,Goal,Provision)
6     : acceptableProvision(Goal,Provision)
7    <- acceptProvision(Agent,Goal,Provision);
8       requestGoal(Agent,Goal).
9
10 +pendingProvision(Agent,Goal,Provision)
11    : not acceptableProvision(Goal,Provision)
12   <- rejectProvision(Agent,Goal,Provision).
13
14 +agreedGoal(Ag,Goal)
15    : acceptedProvision(Ag,Goal,Prov) &
16      Prov
17   <- informProvision(Goal,Prov).
18
19 +Prov
20    : acceptedProvision(Ag,Goal,Prov) &
21      not achievedGoal(Goal)
22   <- informProvision(Goal,Prov).
```

At this point, *companyA* should agree to achieve the *aci* goal to satisfy C_1, given *walls_built*. The agreement will also detach C_2 and will create a commitment $C_3 = C(companyA, Giacomo, inform(done(walls\_built)), inform(done(achieve(aci, walls\_built))))$. Giacomo will, then, execute `informProvision`, thereby satisfying C_2 and detaching C_3. As a final step, *companyA* will achieve the goal by installing the air conditioning (see plan 26 in Listing 3), and will communicate success via `informGoal`. After this step, C_3 will be satisfied, and hopefully the house construction will be successfully completed as expected.

At the same time, in case of organizational goal failure, an investigative entity, possibly Giacomo himself, could inspect the observable properties of the AccountabilityBoard artifact defined during the interaction, which altogether allow one to reconstruct the previously described commitments,

and, on this foundation, discern accountability comprehensively. For instance, should the walls not be built, Giacomo would not be able to inform *companyA* that the previously accepted provision *walls_built* holds. Consequently, *companyA* would not start to pursue the goal because C_3 would not be detached. At the same time, this would cause the violation of commitment C_2, which would correctly lead to considering Giacomo as accountable for the failure, since it was Giacomo who failed to provide an agreed-upon provision. Similarly, should the provision hold and *companyA* not pursue the goal and therefore be unable to preform `informGoal`, the violation of commitment C_3 would allow one to consider the company agent itself accountable. In any case, the investigation process is outside the scope of this work, but the infrastructure provided by the artifact keeps track of the necessary information.

6. Impact and Discussion

MASs have demonstrated to be a viable approach for developing complex, distributed systems. Indeed, several methodologies for designing MASs have been discussed in the literature [1–4,6,7] and, correspondingly, different agent platforms [5,8–11] have been proposed as technological means for practically implementing distributed applications. As discussed in Section 2, most of these design methodologies and platforms adopt the organizational metaphor as a way for distributing and coordinating agents. The organization provides a functional decomposition of organizational goals into subgoals that can be assigned to agents. In addition, norms and obligations arising within an organization specify the good conduct of agents. As we have already pointed out, this approach has been criticized by some recent work [13,16], since such organizations lack a fundamental characterization: the specification of accountability relationships among the agents. In [13], the need for accountability relationships is motivated by the necessity of robustness even in open and dynamic environments. Open systems, in fact, are particularly challenging from the point of view of software engineering, because agents see an organization as a complex resource to be used in order to get their own goals, which may not match with the organizational ones. As shown in [16], the functional decomposition of organizational goals, and their attribution to agents, is not a viable solution for assuring software robustness. In fact, when an agent fails an obligation and does not achieve a goal, it cannot be held accountable since the contextual information is not sufficient to find out the causes, and hence assign the blame. Did the agent have the control over the goal? In other words, did the agent have a plan for reaching the goal? Was it legitimate from the point of view of the organization to expect that the agent had and used such a plan? The impossibility to precisely determine the causes of a failure makes the overall system vulnerable and poorly debugged.

ADOPT provides a mechanism for answering the above questions. By imposing agents to follow a protocol, for enacting a role and for explicitly agreeing to accomplish a goal, our approach makes agents accountable for their activities within the organization. When an agent agrees to achieve a goal, it implicitly declares that is has control over that goal. This is consistent with the fact that agents are not inspectable, and hence only the agent itself knows the tasks it can perform. On the other hand, the agent's declaration allows an organization to expect that the agent will bring about the goal condition. When an agent fails to achieve a goal, thus, the fault can be easily isolated to the agent, if it agreed with the goal, or to the organization, if it assigned that goal to an agent despite the agent not previously accepting the goal. The ADOPT protocol is specified in terms of social commitments. Commitments are indeed a powerful software engineering tool, since they provide a standard for agent correctness. To verify the compliance of an agent to a protocol, in fact, it is sufficient to verify whether the agent is capable of fulfilling the commitments associated with a specific protocol role. This paves the way for methodologies implementing agents driven by the commitments (see for example, the CoSE methodology [63]), and also mechanisms of agent-type checking, in which agent behaviors are typed with the set of commitments they make progress on [64].

Concerning human organizations, business ethics and compliance programs (and, more in general, self-regulatory initiatives) are becoming more and more central [20,21,65,66], bringing to the forefront the importance of accountability. Many organizations and companies voluntarily adopt monitoring

and accountability frameworks, for example, [20,21], or comply to social accountability standards, like [67], which aim at increasing awareness about values (like health and safety, sustainability, no use of child labor, gender equality, and so on.) and also behaviors that account for such values. Principals must be held accountable for their (mis)behavior and, therefore, provide feedback about the reasons of performance.

Accountability frameworks vary considerably, depending on the kind of actors that are involved, on the kind of commitments, and on the activities that may be put under scrutiny. Figure 4 summarizes a general cycle that can be found (declined in many ways) in many organizations. (The picture is inspired by the framework schemas described in [20,22].) An accountability framework relies on further frameworks, not in the picture for simplicity: for monitoring, for risk assessment, and for managing complaint responses. Of these, the monitoring framework is strictly necessary, as monitoring pairs with and completes the obligation to report that is inherent in accountability.

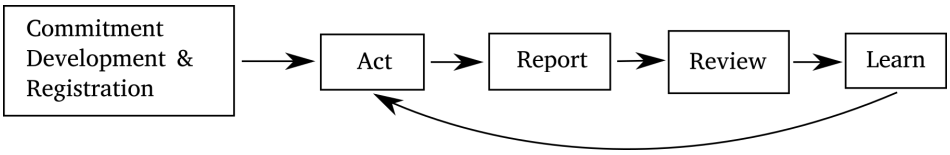

Figure 4. A general scheme for accountability frameworks.

Developing and implementing self-regulatory initiatives is a difficult task which requires human and financial resources. Stronger compliance mechanisms risk reducing participation to a small group of well-resourced organizations. Smaller organizations, often those most in need of support to improve quality and accountability, may be unable to participate [24]. Figure 5 shows the mechanisms through which self-regulation is realized: more often than not, realization of compliance (and its assessment) requires the intervention of external agencies and auditors. Informal structures have a limited impact. Accountability frameworks are currently very-little supported by software and information systems. The commitments that involve the parties are basically hand-written by filling in forms [22,23], and the assessment of satisfaction or violation of the involved liabilities, as well as the actual accounting process, is totally handled by authorized human parties [24]. The problem is that when accountability channels are informal or ambiguous in the long run, all these processes will be thwarted, leading to little effectiveness and poor performance, in particular when cross-organizational relationships must be established and maintained.

The proposal presented in this paper can help in supporting organizations, big and small, in realizing self-regulatory initiatives. Specifically, it allows creating software support to the commitment-creation phase, the monitoring-of-the-act phase, and the identification of the accountable parties who should report about circumstances under scrutiny. The subsequent revision, embodied by the review and learn phases, will instead be carried out by the involved principals. This can be done because the way in which accountability is modeled finds correspondence in the way accountability is understood in human accountability frameworks. Most of these (among which [20,22,23,65,66] share a common interpretation of accountability that is well described by [66] as a relationship between two or more parties that implies responsibilities and consequences) share obligation to take responsibility (belonging to the development of commitments and to the act phase), to demonstrate performance, to review. In particular, taking responsibility belongs to the development of commitments and to the act phase: it entails awareness to answer for what was (not) accomplished and also for the means used in the effort. Expectations about outcomes are intended to be agreed expectations (even in hierarchic contexts), and to stem from either a formal or informal agreement on what should be accomplished.

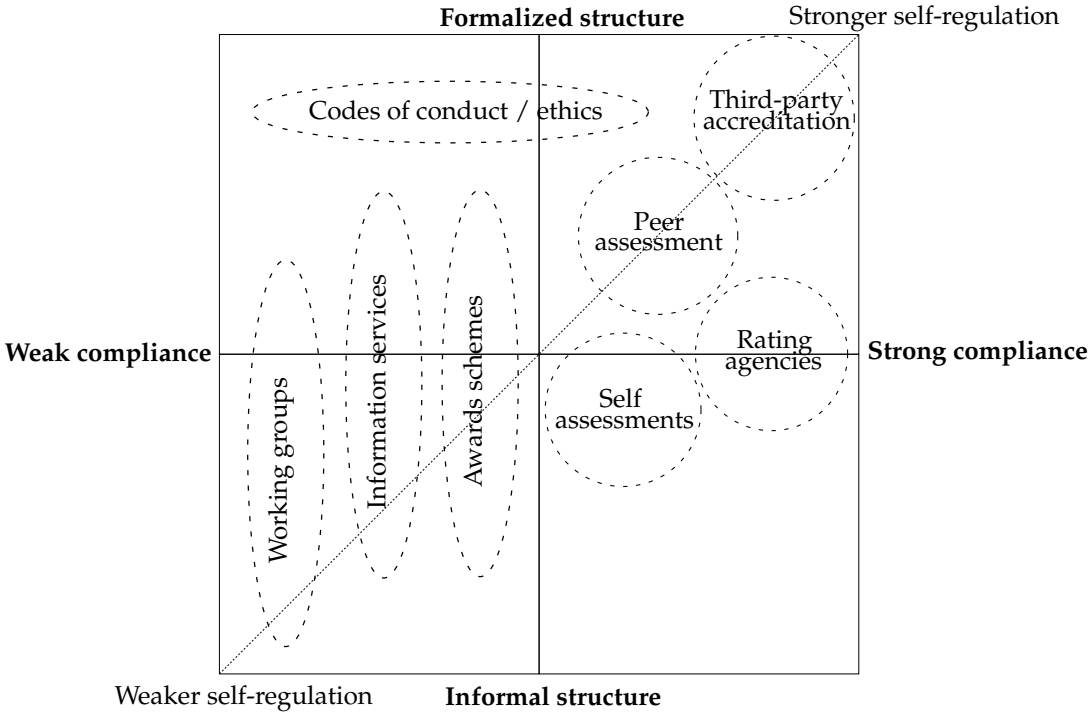

Figure 5. Types of self-regulatory initiatives within individual civil– social organizations [24].

7. Conclusions

Accountability is an important asset in human societies. In this paper we have shown how accountability plays a central role also in MAS engineering. Specifically, we have argued that while MASs rely on the metaphor of (human) organizations as a way for distributing goals and for establishing norms of good behavior, the organizational models in the state of the art are inadequate for developing robust open systems. In open systems, the agents see an organization as a means for reaching their own goals, which may not coincide with the organizational ones. Thus, when an organization assigns a goal to an agent, that organization cannot hold the expectation that the agent will pursue that goal. This poses a question about the robustness of an open MAS, which can only be obtained by relying on the commitments that each agent voluntarily takes towards the organization as a whole. This is the rationale that has driven us in proposing the ADOPT protocol, whose objective is to make agents, and the organization to which they belong, mutually accountable.

We plan to continue this study along two main research directions. The first concerns the specification of a conceptual model of accountability for use in MASs. We believe the notions of expectation and of control to be at the heart of accountability, and find a challenge in capturing the relational nature of this concept. Among the repercussions of such a model lies the possibility to provide organizations with a tool that will enable them to evaluate, and possibly improve, the processes put in place, as well as the whole organizational structure. The second is to account in the model, and also in the protocol, for a notion of negative accountability. This means not only accounting for what agents should have achieved and have not, but also for active interferences, by which agents may impede social progress. In addition, the ADOPT protocol presented in this paper could be complemented with some reasoning mechanism on the forum side. ADOPT, in fact, aims at tracing, during the interaction, the relevant information for the computation of accountability, but it does not address how to use such information to identify who is accountable. Indeed, this is the task of the forum [38]. A possible way to support the forum, thus, could be represented by the Delphi protocol, whose first formalization in terms of a MAS organization is given in [68] by means of the INGENIAS [69] methodology. In the Delphi protocol, a client submits to a group of experts a question, and one, or more, monitor agents set up an iterative interaction among the experts until a consensus

is reached about the answer to the question. A similar mechanisms for consensus reaching can be adopted for automating, at least in part, the task of the forum.

Supplementary Materials: The source code of the AccountabilityBoard artifact and of the examples presented in Section 5 is available at http://di.unito.it/adoptjacamo.

Acknowledgments: This work was partially supported by the *Accountable Trustworthy Organizations and Systems (AThOS)* project, funded by Università degli Studi di Torino and Compagnia di San Paolo (CSP 2014).

Author Contributions: All authors equally contributed to this research.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CNP	Contract Net Protocol
CTL	Computation tree logic
CTLC	Computation tree logic with commitment modality
CTLK	Computation tree logic with epistemic modality
FIPA	Foundation for Intelligent Physical Agents
IOSE	Interaction-oriented software engineering
MAS	Multiagent system
STS	Socio–technical system

References

1. Esteva, M.; Rodríguez-Aguilar, J.A.; Sierra, C.; Garcia, P.; Arcos, J.L. On the Formal Specification of Electronic Institutions. In *Agent Mediated Electronic Commerce: The European AgentLink Perspective*; Dignum, F., Sierra, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2001; pp. 126–147.
2. Dignum, V. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*; SIKS: Lyon, France, 2004.
3. Dignum, V.; Vázquez-Salceda, J.; Dignum, F. *OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations*; Lecture Notes in Artificial Intelligence; Springer: Berlin/Heidelberg, Germany, 2004; Volume 4, pp. 181–198.
4. Fornara, N.; Viganò, F.; Verdicchio, M.; Colombetti, M. Artificial institutions: A model of institutional reality for open multiagent systems. *Artif. Intell. Law* **2008**, *16*, 89–105.
5. Mariani, S.; Omicini, A. Coordinating activities and change: An event-driven architecture for situated MAS. *Eng. Appl. Artif. Intell.* **2015**, *41*, 298–309.
6. Zambonelli, F.; Jennings, N.R.; Wooldridge, M. Developing multiagent systems: The Gaia methodology. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2003**, *12*, 317–370.
7. Kolp, M.; Giorgini, P.; Mylopoulos, J. Multi-agent architectures as organizational structures. *Auton. Agents Multi-Agent Syst.* **2006**, *13*, 3–25.
8. Bordini, R.H.; Hübner, J.F.; Wooldridge, M. *Programming Multi-Agent Systems in AgentSpeak Using Jason*; John Wiley & Sons: Hoboken, NJ, USA, 2007; Volume 8.
9. Ricci, A.; Piunti, M.; Viroli, M.; Omicini, A. Environment Programming in CArtAgO. In *Multi-Agent Programming: Languages, Tools and Applications*; Springer: Boston, MA, USA, 2009; pp. 259–288.
10. Dastani, M.; Tinnemeier, N.A.; Meyer, J.J.C. A programming language for normative multiagent systems. In *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*; IGI Global: Hershey, PA, USA, 2009; pp. 397–417.
11. Boissier, O.; Bordini, R.H.; Hübner, J.F.; Ricci, A.; Santi, A. Multi-agent Oriented Programming with JaCaMo. *Sci. Comput. Program.* **2013**, *78*, 747–761.
12. Wooldridge, M.J. *Introduction to Multiagent Systems*; Wiley: Hoboken, NJ, USA, 2009.
13. Chopra, A.K.; Singh, M.P. From social machines to social protocols: Software engineering foundations for sociotechnical systems. In Proceedings of the 25th International Conference on World Wide Web, Montréal, QC, Canada, 11–15 April 2016.

14. Baldoni, M.; Baroglio, C.; May, K.M.; Micalizio, R.; Tedeschi, S. Computational Accountability. In Proceedings of the AI*IA Workshop on Deep Understanding and Reasoning: A challenge for Next-Generation Intelligent Agents, URANIA 2016, Genova, Italy, 28 November 2016; Chesani, F., Mello, P., Milano, M., Eds.; Volume 1802, pp. 56–62.
15. Baldoni, M.; Baroglio, C.; Capuzzimati, F.; Micalizio, R. Commitment-based Agent Interaction in JaCaMo+. *Fundam. Inform.* **2018**, *159*, 1–33.
16. Baldoni, M.; Baroglio, C.; May, K.M.; Micalizio, R.; Tedeschi, S. ADOPT JaCaMo: Accountability-Driven Organization Programming Technique for JaCaMo. In *PRIMA 2017: Principles and Practice of Multi-Agent Systems*; Bo, A., Bazzan, A., Leite, J., van der Torre, L., Villata, S., Eds.; Lecture Notes in Artificial Intelligence; Springer: Nice, France, 2017; Volume 10621, pp. 295–312.
17. Anderson, P.A. Justifications and Precedents as Constraints in Foreign Policy Decision-Making. *Am. J. Political Sci.* **1981**, *25*, 738–761.
18. Castelfranchi, C. Commitments: From Individual Intentions to Groups and Organizations. In Proceedings of the First International Conference on Multiagent Systems, ICMAS 1995, San Francisco, CA, USA, 12–14 June 1995; Lesser, V. R., Gasser, L., Eds.; pp. 41–48.
19. Singh, M.P. An ontology for commitments in multiagent systems. *Artif. Intell. Law* **1999**, *7*, 97–113.
20. Zahran, M. *Accountability Frameworks in the United Nations System*; UN Report A/66/710; United Nations: New York, NY, USA, 2012. Available online: http://repository.un.org/handle/11176/293914 (accessed on 22 March 2018).
21. United Nations Children's Fund. *Report on the Accountability System of UNICEF*; E/ICEF/2009/15; UNICEF: New York, NY, USA, 2009. Available online: https://www.unicef.org/about/execboard/files/09-15-accountability-ODS-English.pdf (accessed on 22 March 2018).
22. Sustainable Energy for All Initiative. *Accountability Framework*; United Nations: New York, NY, USA, 2014. Available online: https://sustainabledevelopment.un.org/content/documents/1644se4all.pdf (accessed on 22 March 2018).
23. Obrecht, A. *Effective Accountability? The Drivers, Benefits and Mechanisms of CSO Self-Regulation*; Technical Report Briefing No. 130; One World Trust: London, UK, 2012.
24. Warren, S.; Lloyd, R. *Civil Society Self-Regulation*; Technical Report Briefing Paper Number 119; One World Trust: London, UK, 2009.
25. Baldoni, M.; Boella, G.; Genovese, V.; Mugnaini, A.; Grenna, R.; van der Torre, L. A Middleware for Modelling Organizations and Roles in Jade. In Proceedings of the Programming Multi-Agent Systems (ProMAS 2009), Budapest, Hungary, 10–15 May 2009; Braubach, L., Briot, J.P., Thangarajah, J., Eds.; Lecture Notes in Artificial Intelligence; Springer: Berlin/Heidelberg, Germany, 2010; Volume 5919, pp. 100–117.
26. Boella, G.; van der Torre, L. The ontological properties of social roles in multiagent systems: Definitional dependence, powers and roles playing roles. *Artif. Intell. Law* **2007**, *15*, 201–221.
27. Chopra, A.K.; Dalpiaz, F.; Aydemir, F.B.; Giorgini, P.; Mylopoulos, J.; Singh, M.P. Protos: Foundations for engineering innovative sociotechnical systems. In Proceedings of the IEEE 22nd International Requirements Engineering Conference (RE 2014), Karlskrona, Sweden, 25–29 August 2014; pp. 53–62.
28. Bella, G.; Paulson, L.C. Accountability Protocols: Formalized and Verified. *ACM Trans. Inf. Syst. Secur.* **2006**, *9*, 138–161.
29. De Oliveira, A.S.; Charfi, A.; Schmeling, B.; Serme, G. A Model-Driven Approach for Accountability in Business Processes. In *Enterprise, Business-Process and Information Systems Modeling*; Bider, I., Gaaloul, K., Krogstie, J., Nurcan, S., Proper, H.A., Schmidt, R., Soffer, P., Eds.; Lecture Notes in Business Information Processing; Springer: Berlin/Heidelberg, Germany, 2014; Volume 175, pp. 184–199.
30. Yumerefendi, A.R.; Chase, J.S. The Role of Accountability in Dependable Distributed Systems. In Proceedings of the First Conference on Hot Topics in System Dependability, Yokohama, Japan, 28 June–1 July 2005; USENIX Association: Berkeley, CA, USA, 2005; p. 3.
31. Haeberlen, A.; Kouznetsov, P.; Druschel, P. PeerReview: Practical Accountability for Distributed Systems. *SIGOPS Oper. Syst. Rev.* **2007**, *41*, 175–188.
32. Kramer, S.; Rybalchenko, A. A Multi-Modal Framework for Achieving Accountability in Multi-Agent Systems. In Proceedins of the Workshop on Logics in Security, Copenhagen, Denmark, 9–13 August 2010; pp. 148–174.
33. Nissenbaum, H. Accountability in a computerized society. *Sci. Eng. Ethics* **1996**, *2*, 25–42.

34. Mao, W.; Gratch, J. Modeling Social Causality and Responsibility Judgment in Multi-agent Interactions. *J. Artif. Intell. Res.* **2012**, *44*, 223–273.
35. Feltus, C.; Petit, M. Building a Responsibility Model Including Accountability, Capability and Commitment. In Proceedings of the International Conference on Availability, Reliability and Security, Fukuoka, Japan, 16–19 March 2009; pp. 412–419.
36. Küsters, R.; Truderung, T.; Vogt, A. Accountability: Definition and Relationship to Verifiability. In Proceedings of the 17th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 4–8 October 2010; ACM: New York, NY, USA, 2010; pp. 526–535.
37. Bovens, M.; Goodin, R.E.; Schillemans, T. *The Oxford Handbook of Public Accountability*; Oxford University Press: Oxford, UK, 2014.
38. Burgemeestre, B.; Hulstijn, J. Designing for Accountability and Transparency: A value-based argumentation approach. In *Handbook of Ethics, Values, and Technological Design: Sources, Theory, Values and Application Domains*; Springer: Basel, Switzerland, 2015.
39. Simon, J. *The Online Manifesto: Being human in a Hyperconnected Era*; Floridi, L., Ed.; Springer: Basel, Switzerland, 2015.
40. Chopra, A.K.; Singh, M.P. The thing itself speaks: Accountability as a foundation for requirements in sociotechnical systems. In Proceedings of the IEEE 7th International Workshop on Requirements Engineering and Law (RELAW), Karlskrona, Sweden, 26 August 2014; p. 22.
41. Braham, M.; van Hees, M. An Anatomy of Moral Responsibility. *Mind* **2012**, *121*, 601–634.
42. Marengo, E.; Baldoni, M.; Baroglio, C.; Chopra, A.K.; Patti, V.; Singh, M.P. Commitments with Regulations: Reasoning about Safety and Control in REGULA. In Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, 2–6 May 2011; Tumer, K., Yolum, P., Sonenberg, L., Stone, P., Eds.; Volume 2, pp. 467–474.
43. Dastani, M.; Lorini, E.; Meyer, J.C.; Pankov, A. Other-Condemning Anger = Blaming Accountable Agents for Unattainable Desires. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, São Paulo, Brazil, 8–12 May 2017; pp. 1520–1522.
44. Hohfeld, W.N. Some Fundamental Legal Conceptions as Applied in Judicial Reasoning. *Yale Law J.* **1913**, *23*, 16–59.
45. Foundation for Intelligent Physical Agents. *FIPA ACL Message Structure Specification*; Foundation for Intelligent Physical Agents: Geneva, Switzerland, 2002.
46. Castelfranchi, C. *Commitments: From Individual Intentions to Groups and Organizations*; The MIT Press: Cambridge, MA, USA, 1995; pp. 41–48.
47. Telang, P.R.; Singh, M.P.; Yorke-Smith, N. *Relating Goal and Commitment Semantics*; Lecture Notes in Artificial Intelligence; Springer: Berlin/Heidelberg, Germany, 2011; Volume 7217, pp. 22–37.
48. Smith, R.G. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Trans. Comput.* **1980**, *29*, 1104–1113.
49. Foundation for Intelligent Physical Agents. *FIPA Contract Net Interaction Protocol Specification*; Foundation for Intelligent Physical Agents: Geneva, Switzerland, 2002.
50. Foundation for Intelligent Physical Agents. *FIPA Request Interaction Protocol Specification*; Foundation for Intelligent Physical Agents: Geneva, Switzerland, 2002.
51. Lai, R.; Jirachiefpattana, A. Protocol Verification. In *Communication Protocol Specification and Verification*; Springer: Boston, MA, USA, 1998; pp. 143–163.
52. Alpern, B.; Schneider, F.B. Recognizing safety and liveness. *Distrib. Comput.* **1987**, *2*, 117–126.
53. Sajkowski, M. Protocol Verification Techniques: Status Quo and Perspectives. Protocol Specification, Testing and Verification IV. In Proceedings of the IFIP WG6.1 Fourth International Workshop on Protocol Specification, Testing and Verification, Skytop Lodge, PA, USA, 11–14 June 1984; pp. 697–720.
54. El-Menshawy, M.; Bentahar, J.; Dssouli, R. Symbolic model checking commitment protocols using reduction. In Proceedings of the 8th International Workshop on Declarative Agent Languages and Technologies, DALT 2010, Toronto, ON, Canada, 10 May 2010; Lecture Notes in Artificial Intelligence; Volume 6619, pp. 185–203.
55. Clarke, E.M.; Emerson, E.A.; Sistla, A.P. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.* **1986**, *8*, 244–263.

56. Penczek, W.; Lomuscio, A. Verifying Epistemic Properties of Multi-agent Systems via Bounded Model Checking. *Fundam. Inform.* **2003**, *55*, 167–185.
57. Lomuscio, A.; Qu, H.; Raimondi, F. MCMAS: An open-source model checker for the verification of multiagent systems. *Int. J. Softw. Tools Technol. Transf.* **2017**, *19*, 9–30.
58. Hubner, J.F.; Sichman, J.S.; Boissier, O. Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels. *Int. J. Agent-Oriented Softw. Eng.* **2007**, *1*, 370–395.
59. Rao, A.S. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away*; Lecture Notes in Artificial Intelligence; Springer: Berlin/Heidelberg, Germany, 1996; Volume 1038, pp. 42–55.
60. Omicini, A.; Ricci, A.; Viroli, M. Artifacts in the A&A meta-model for multiagent systems. *Auton. Agents Multi-Agent Syst.* **2008**, *17*, 432–456.
61. Weyns, D.; Omicini, A.; Odell, J. Environment as a first class abstraction in multiagent systems. *Auton. Agents Multi-Agent Syst.* **2007**, *14*, 5–30.
62. Hübner, J.F.; Boissier, O.; Kitio, R.; Ricci, A. Instrumenting multiagent organisations with organisational artifacts and agents. *Auton. Agents Multi-Agent Syst.* **2010**, *20*, 369–400.
63. Baldoni, M.; Baroglio, C.; Capuzzimati, F.; Micalizio, R. Empowering Agent Coordination with Social Engagement. In *AI*IA 2015, Advances in Artificial Intelligence*; Lecture Notes in Artificial Intelligence; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9336, pp. 89–101.
64. Baldoni, M.; Baroglio, C.; Capuzzimati, F.; Micalizio, R. Type checking for protocol role enactments via commitments. *Auton. Agents Multi-Agent Syst.* **2018**, 1–38, doi:10.1007/s10458-018-9382-3.
65. World Health Organization. WHO Accountability Framework. 2015. Available online: http://www.who.int/about/who_reform/managerial/accountability-framework.pdf (accessed on 22 March 2018).
66. Office of the Auditor General of Canada. Modernizing Accountability in the Public Sector. In *Report of the Auditor General of Canada*; Minister of Public Works and Government Services Canada: Ottawa, ON, Canada, 2002; Chapter 9. Available online: http://www.oag-bvg.gc.ca/internet/English/parl_oag_200212_09_e_12403.html (accessed on 22 March 2018).
67. Social Accountability International. *Social Accountability 8000 International Standard*; Social Accountability International: New York, NY, USA, 2014.
68. García-Magariño, I.; Gómez-Sanz, J.J.; Pérez-Agüera, J.R. A multiagent based implementation of a Delphi process. In Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, 12–16 May 2008; Volume 3, pp. 1543–1546.
69. Gómez-Sanz, J.J.; Pavón, J. Implementing Multi-agent Systems Organizations with INGENIAS. In Proceedings of the Third International Workshop on Programming Multi-Agent Systems (ProMAS 2005), Utrecht, The Netherlands, 26 July 2005; Lecture Notes in Artificial Intelligence; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3862, pp. 236–251.