Robustness based on Accountability in Multiagent Organizations

Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi
Università di Torino, Dipartimento di Informatica, Torino, Italy
firstname.surname@unito.it

ABSTRACT

Multiagent system models do not typically encompass tools for tackling abnormal or exceptional situations. When a critical situation arises, and individual agents fail, the system as a whole usually fails, too. For realizing robust distributed systems, conceived as agent systems or organizations, it is necessary to keep a right level of situational awareness (of both agents and environment), through the introduction of the means for gathering and propagating feedback, upon which actions can be taken. This work proposes a notion of accountability that encompasses both a normative dimension, and a structural dimension, that serves for the purpose of robustness.

KEYWORDS

Accountability; Responsibility; Agent Organizations; Engineering MAS; Robustness

ACM Reference Format:
Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi. 2021. Robustness based on Accountability in Multiagent Organizations. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

Agent organizations (e.g., [2]) are a well-known abstraction for conceptualizing and developing distributed systems. The organization metaphor is, in fact, a useful mechanism for modularizing code spread over different software components that are opaque and independent of each other. Agent organizations show the same kind of structure and of advantage that sociologist Dave Elder-Vass explains for human organizations: an organization provides a structure of constraints that allow a system consisting of many parts to act as a whole, with the aim of achieving goals that otherwise would not be achievable (or not as easily) [25].

Many approaches to multiagent systems and organizations, e.g. [10, 14, 20, 32], provide the means to design and realize the correct, expected behavior of the system, capturing exactly what agents should do to contribute to the achievement of the organizational goal. For instance, many approaches rely on *norms* (rules, protocols, etc.) to define what is expected of each agent and which sanction is applied when an agent does not comply. Sanctions are intended as deterrents to prevent norm violation, i.e., to keep the execution oriented towards the achievement of the organizational goal.

The problem is that when the system faces an abnormal situation (i.e., a *perturbation*) and some agent *fails* to achieve a goal, sanctions are of little utility, if any [6, 9, 16]. In this case, in fact, the agent may have earnestly tried its best to do what expected, but something which is not under its control hindered the achievement. To have an intuition, an agent may fail to deliver a parcel because a tree that fell blocks the only way that allows to reach the recipient.

What is missing in the picture is some support for allowing agents to provide an *account* on what happened, propagating it through appropriately devised structure inside the organization, for reaching those agents that are equipped with the means for coping with them. Such a tool aims at making the organization more *robust*, that is, capable to keep an acceptable behavior in spite of unforeseen, abnormal, or stressful conditions. We see *robustness* as a crucial property for scaling from agents to agent organizations, and make a proposal that is based on the concepts of *accountability* and *responsibility*.

Accountability is extremely important in the human world. The kind of accountability we refer to is well-described in a report by the United Nations Development Programme (UNDP) [26]. UNDP's accountability framework describes organization-wide processes for monitoring, analyzing, and improving performance in all aspects of the organization. The framework gives managers the means to address recurring and systemic issues, and to incorporate lessons learned into future activities. Inside the framework, accountability is supported, among other things, by formally documented functions, responsibilities, authority, management expectations, policies, processes and instruments for improving performance.

Contribution. The main contribution of this paper is a formal treatment of robustness in multiagent organizations (MAOs), that permits the specification of a proper treatment of possible perturbations. The proposal is independent of the specific organizational model. We introduce the notion of accountability, along both its *normative* and its *structural* dimensions, to express who is expected to provide feedback to whom (and under which circumstances), and to specify further requirements on how responsibilities should be distributed in the system. Based upon accountability, we present a formal treatment of robustness by introducing some key definitions and properties, and by showing how accountability supports the system to reaching a consistent state (an acceptable termination point) despite perturbations. We illustrate the practical use of the proposal in JaCaMo [12]. Even though we focus on abnormal situations, the proposal can be extended to *reports* concerning behavior in positive cases, a feature that supports compliance to standards and transparency [8, 9].

2 WORKFLOWS: BACKGROUND

Precedence logic [41], supplies the means for representing possibily complex conditions, that are concerned in accountability. Such conditions will amount to *workflows* to which agents participate,

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

and that are expected to take place. In general, precedence logic defines (temporal) expressions that are suitable for describing possible execution runs of (distributed) system, and relies on the notion of *residuation* for tracking the progression of these runs upon the occurrence of system events. The logic has three primary operators: '$\vee$' (choice), '$\wedge$' (interleaving), and '$\cdot$' (ordering). *Ordering* allows constraining the order with which two events must occur, e.g., $a \cdot b$ means that a must occur before b, but the two events do not need to occur one immediately after the other. Instead, *choice* specifies that at least one of the events should occur, while *interleaving* that all should occur but the order is unimportant. The *residual* of a workflow u with respect to an event e, denoted as u/e, is the remainder workflow that would be left over when e occurs, and whose satisfaction would guarantee the satisfaction of the workflow u. Residual can be calculated by means of a set of rewrite rules. The following equations are due to Singh (2003). Here, u is a workflow, e is an event or $\top$, and $\overline{e}$, the complement of e, is also an event. Initially, neither e nor $\overline{e}$ hold. On any run, either e or $\overline{e}$ may occur but not both. Note that we assume that events are nonrepeating. In practice, we can assume that timestamps differentiate multiple instances of the same event. Below, Γ_u is the set of literals and their complements mentioned in u. Thus, for instance, $\Gamma_e = \{e, \overline{e}\} = \Gamma_{\overline{e}}$ and $\Gamma_{e \cdot f} = \{e, \overline{e}, f, \overline{f}\}$.

$$\begin{array}{ll}
0/e \doteq 0 & \top/e \doteq \top \\
(r \wedge u)/e \doteq ((r/e) \wedge (u/e)) & (r \vee u)/e \doteq ((r/e) \vee (u/e)) \\
(e \cdot r)/e \doteq r, \text{ if } e \notin \Gamma_r & r/e \doteq r, \text{ if } e \notin \Gamma_r \\
(e' \cdot r)/e \doteq 0, \text{ if } e \in \Gamma_r & (\overline{e} \cdot r)/e \doteq 0
\end{array}$$

An event e is *relevant* to a workflow u if that event is involved in u, i.e. $u/e \not\equiv u$ [5]. We use the expression $r/(e_1, \ldots, e_n)$ as a shortcut for $((r/e_1)/\ldots)/e_n$. Finally, let $w = (e_1, \ldots, e_k)$ and $z = (e_{k+1}, \ldots, e_n)$ be two sequences of events, wz is their concatenation $(e_1, \ldots, e_k, e_{k+1}, \ldots, e_n)$. We denote by $u[r]$ the fact that the workflow u contains the sub-workflow r. For example, if $u = a \wedge (b \cdot c)$, we can write $u[b \cdot c]$ because of $b \cdot c$ is contained in u. Of course, we have that $\forall u : u[u]$; i.e., a workflow is trivially a sub-workflow of itself. Moreover, we denote by $u \longrightarrow u'$ the fact that for any $(e_1, \ldots, e_n)$ such that $u/(e_1, \ldots, e_n) = \top$ we also have that $u'/(e_1, \ldots, e_n) = \top$.

Example 1 (Yoghurt production line). Let us consider an automated yoghurt production line: a robotic arm takes a cup from the storage and places it on a conveyor belt while a filler is placed in correspondence to the final position of the cup. The cup is, then, filled and finally the lid is put on it. This workflow can be modeled as: production =(takeCup $\wedge$ enableFiller) $\cdot$ fillCup $\cdot$ putLid.

3 A MODEL OF ROBUST MAO

The notion of accountability has recently gained the attention of many authors (see e.g., [3, 16, 19]), who see a powerful software engineering tool in it. We agree and add that accountability is fundamental to design and realize robust, agent systems. Alderson and Doyle (2010) say on robustness: "*A [property] of a [system] is robust if it is [invariant] with respect to a [set of perturbations].*" Brackets are original and emphasize that a formal treatment of robustness requires a proper system specification. In other terms, robustness is primarily a matter of *good design*, which in turn demands for proper engineering tools. We see in accountability such a tool. Taking as references the conceptual models of organizations and institutions discussed in [12, 21–23, 27, 32, 43], we have distilled a minimal conceptual model, in Figure 1, where white boxes represent the concepts commonly underpinning social models.

3.1 Norms, Roles, Responsibilities, Sanctions, Tasks

Following the cited works, where the society is shaped by a set of norms, that create social expectations through, e.g., commitments, authorizations, prohibitions [42], we introduce the concept of `Norm`. Norms can yield obligations about the tasks that agents are held to fulfill; they are, therefore, used to describe the ideal behavior of agents in terms of their responsibilities, rights and duties [34]. Many methodologies for agent organizations (e.g., [22, 32, 43]) hinge on the concepts of `Task` and of `Responsibility` about some `Task`. So, in Gaia [43], the functionality of a role is defined by its responsibilities. The OperA framework [22] is able to define the global aims of an organization (tasks), and the objectives and responsibilities of its participants. A similar idea recurs in other frameworks, such as OMNI [23] and MOISE [32], where a functional decomposition describes how a complex goal (task) can be achieved in a distributed way. Agents joining the organization are expected to contribute by achieving subgoals of such a decomposition, whenever obligations are triggered by the organization toward them. In MOISE, agents are held to explicitly commit to missions (i.e., subsets of goals), this act implies an assumption of responsibility of the agents toward their missions and, hence, the acceptance of the related obligations that will be issued by the organization.

In short, all these frameworks see responsibility as duties that an agent has to accomplish while in an organization, and use obligations as a mechanism for telling agents how to discharge their responsibilities, accomplishing their tasks. The agent autonomy is preserved, since agents can reason about the normative system, and hence can deliberate how to act to trigger obligations, or they can even decide not to satisfy an obligation despite being possibly sanctioned. Thus, norms originate responsibilities in organizations [27], as well as in institutions [34], and responsibilities aggregate the tasks that can be performed within the society. Norms are also associated with `Sanction`, that can be imposed to agents when they violate some norm. A `Role` models, through norms, the powers and duties of its players. An `Agent` can adopt/leave a `Role`, commit/leave a `Responsibility`, and can achieve/fail a `Task` by internalizing it as a goal of its own. We assume agents to be autonomous, thus, we require them to explicitly *commit to their responsibilities*.

In the following, we model complex tasks as workflows, or processes, expressed in precedence logic (see Section 2), and denote as $\mathsf{R}(x, u)$ that x has the responsibility for task u. By way of $\mathsf{R}(\cdot)$ it is possible to *impose requirements* on the distribution of responsibilities among the agents that will constitute an organization. We assume that each agent, taking part to a workflow, takes on the responsibility for the portion of the same workflow that it is expected to execute. We assume responsibility to be invariant with respect to the occurrence of events. $\mathbf{R}_u$ denotes a distribution of responsibility over a workflow u. It amounts to a set of responsibilities of form $\mathsf{R}(x, u')$, such that $u[u']$.

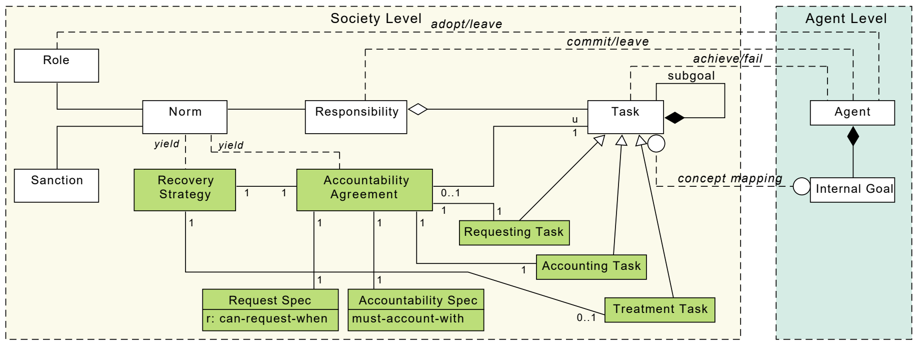

Figure 1: The proposed conceptual model of robust multi-agent organizations.

Definition 1 (Simplest responsibility coverage). *Given a workflow u, we say that a responsibility distribution $\mathbf{R}_u$ is a simplest responsibility coverage of u iff for each u' such that $u[u']$ and u' is atomic there exists $r \in \mathbf{R}_u$ such that $r = \mathsf{R}(x, u')$ for some x.*

Intuitively, a simplest responsibility coverage over u assures that there is some agent who is in charge of each atomic step in u.

3.2 Recovery Strategies, Accountability Agreements, Request and Account Specs

In the proposed model, robustness relies on the concept of `Recovery Strategy`. Robustness is obtained by making the system tolerant to perturbations for which an adequate treatment is known. A `Recovery Strategy` connects the account for a perturbation with a `Treatment Task` that can properly tackle it. Note that `Treatment Task` may be optional in a recovery strategy. The rationale is that it may not be necessary to treat all perturbations: in some situations, we just want to raise awareness of it. Since a `Treament Task` is a `Task`, its related `Responsibility` can be assinged to a `Role`. Consequently, obligations can be created concerning its execution.

Building upon sociology literature [24, 28, 39], in agent organizations (e.g., [3, 19]), accountability is often seen as a mutually accepted social relationship between two parties such that (1) one of the parties ("account taker" or *a-taker*) can legitimately demand, under certain conditions, an account about a process of interest, and (2) the other ("account giver" or *a-giver*) is legitimately required to provide the account. In accountability, we recognize two dimensions:

- The *normative* dimension creates mutual expectations on the behavior of the involved agents; it captures the *legitimacy*, for the account taker, of asking (and the *availability* of the account giver to provide) an account (the standing of the account taker to demand an account).
- The *structural* dimension concerns the *capability* to produce an account; for being held to account about a process, an agent must exert control over the same process and must have proper awareness of the situation it accounts for, possibly by relying on other agents.

We will use the expression *accountability agreement* for talking about the normative dimension, while *accountability structure* will refer to the structural dimension while *accountability* will encompass both dimensions. The former is captured by the concept `Accountability Agreement`, while the latter, together with the complete definition of accountability, will be introduced in Section 4.1 by specifying structural properties that must hold.

Formally, we represent an `Accountability Agreement` as $\mathbb{A}(x, y, r, u)$. Here, x and y are agents, while r and u are expressed in precedence logic because they refer to workflows of tasks, that are expected to occur during the interaction due to some social norms. When r holds, two conditions should be implied: (1) y has the claim-right to ask x for an account about u; and (2) x is actually in condition to provide substantive and authoritative accounts about u. This happens when x can retrieve the necessary contextual information about u, either because the agent is directly responsible for the execution of u, or because it can rely on other agents to gather feedback on the execution of (all the parts of) u. While the first condition is implied by the normative dimension of the accountability, the second condition is only achieved by imposing an adequate structure to the set of defined accountability agreements. Note that an `Accountability Agreement` does not imply that x will bring about u, but only that x can produce an account about u, possibly by relying on accounts provided by other agents.

The association between `Accountability Agreement` and `Task` captures the object of the account. That is, the a-giver is expected to produce an account that is relevant for the task indicated via this association. In our formal representation of an agreement, this corresponds to the process u, for which x is capable of producing accounts, as discussed.

`RequestSpec` specifies the kind of the request the a-taker can make. It is characterized by a `can-request-when` field (mapping condition r), that specifies when an account request is legitimate, i.e., when the a-taker has permission of asking for an account. The association between `Accountability Agreement` and `Requesting Task` specifies who will serve the purpose of being a-taker, that is, the one who is responsible for the `Requesting Task`. Note that, in some cases, an account request is the result of an internal deliberation of the a-taker: asking for an account is just a local goal of the a-taker, and has not a corresponding task at the social level. In other circumstances, instead, an agent might have the duty (i.e., an obligation) to handle a situation of interest, and hence to ask for an account about that condition.

The concept `AccountSpec`, characterized by a `must-account-with` field, captures the type of knowledge that the a-giver feeds back to the a-taker upon request. The association between `Accountability Agreement` and `Accounting Task` specifies who will serve the purpose of being a-giver, that is, the one who is responsible for the `Accounting Task`.

We assume that accountability agreements are preserved at runtime. Given $\mathbb{A}(x, y, r, u)$, we define its progression against the occurrence of an event e as $\mathbb{A}(x, y, r, u)/e = \mathbb{A}(x, y, r/e, u)$. When r/e progresses to $\top$, y is legitimated to ask x for an account, and x is required upon request, to provide y with an account of u. On the contrary, when r/e progresses to 0, the accountability agreement is withdrawn. Notably, events make progress only r, whereas u, the object of the account, remains unchanged. The point is that r determines when an account request is legitimate (or even no longer possible). On the other hand, u, which is independent of r, determines the set of possible accounts, including errors. The two conditions, thus, have a substantially different nature.

Finally, accountability agreements and recovery strategies bring along normative expectations that can be formulated according to the normative layer (`Norm`). So, for instance, they can be grafted both on top of approaches that rely on "permissions" and "obligations", like [11], and on top of frameworks that exploit social commitments [16, 40]. To our aims, it is not necessary to make any assumption on the selected realization, so we will stick to the abstraction level of accountability agreements. Thus, we add at conceptual level a dashed link between `Accountability Agreement` and `Norm`, to underline that there exists a tight bound between agreements and norms, but leave the choice of how to concretize agreements at the implementation level.

4 ROBUSTNESS UPON ACCOUNTABILITY

Rephrasing Alderson and Doyle (2010), a process u is robust to some perturbation, when u includes some kind of recovery that allows u to terminate smoothly despite the occurrence of the perturbation (i.e., to terminate leaving the system in a "consistent" state). We will see that, in order to allow recovery, agents will produce and return accounts that witness what happened to disrupt the execution – e.g., the complement of some expected event occurred, or two events occurred in the wrong order.

Definition 2 (Account). *Given a workflow u:*

- *An account for u is a sequence of events $\widetilde{u} = (e_1, e_2, \ldots, e_n)$, such that for each e_i, $u/e_i \neq u$ (the event e_i is relevant for u).*
- *An account $\widetilde{u_!} = (e_1, e_2, \ldots, e_n)$ witnesses a perturbation for u when $u/(e_1, e_2, \ldots, e_n) = 0$.*

We denote as $\overleftarrow{u_!}$ a recovery event based on the account $\widetilde{u_!}$.

With reference to the model, $\overleftarrow{u_!}$ amounts to the outcome of the `Treatment Task`, associated with the `Recovery Strategy`. Note that an account is not the perturbation it witnesses; if needed, perturbations could be identified by approaches like [36].

Robustness is a property that a system has, or has not. It concerns a perturbation and requires the existence of appropriate *handlers*, i.e., recovery strategies foreseen at an organizational level, which the agent, receiving the perturbation account, should activate for bringing the system to an acceptable state. Specifically, Proposition 1 (below) states that a workflow is robust to a perturbation, when it includes a recovery strategy, that can deal with the perturbation account so as to bring the workflow to an acceptable termination state. Formally, a *recovery strategy* for a workflow u for a perturbation with account $\widetilde{u_!} = (e_1, \ldots, e_n)$ is the workflow $h(\widetilde{u_!}) = e_1 \cdot \ldots \cdot e_n \cdot \overleftarrow{u_!}$. If the workflow u is perturbed with the account $\widetilde{u}$, the workflow $u \vee h(\widetilde{u_!})$ is not. More importantly, $u \vee h(\widetilde{u_!})/(\widetilde{u_!}, \overleftarrow{u_!}) = \top$; that is, when a perturbation is followed by an account describing its context and a suitable recovery event, the system terminates in a consistent state. More generally, we can prove the following proposition.

Proposition 1 (Robustness to a perturbation). *Let u be a workflow such that $u[v]$, where v is perturbed with the account $\widetilde{v_!}$. Let w and z, be two sequences of events, such that $u/wtz = \top$ and $v/t = \top$, for some sequence t. We have that $(u \vee h(\widetilde{v_!}))/wt'z = \top$, where $t' = (\widetilde{v_!}, \overleftarrow{v_!})$.*

Proof. The proof is by induction on the structure of u.

If $u \equiv v$, then w and z are empty, and we just need to prove that $(v \vee h(\widetilde{v_!}))/t' = \top$. This, however, follows directly from the definition of recovery strategy since $h(\widetilde{v_!}) = e_1 \cdot \ldots \cdot e_n \cdot \overleftarrow{v_!} = \widetilde{v_!} \cdot \overleftarrow{v_!}$, and $t' = (\widetilde{v_!}, \overleftarrow{v_!})$ by hypothesis. Thus, $(v \vee h(\widetilde{v_!}))/t' = v/t' \vee h(\widetilde{v_!})/t'$; the first disjunct progresses to 0 whereas the second to $\top$.

In the general case, $v \not\equiv u$, and $u[v]$. Suppose that $u = a \wedge u'$ and $u'[v]$. By inductive hypothesis $(u' \vee h(\widetilde{v_!}))/w't'z = \top$, where $w = aw'$. By progression, we have $(a \wedge u' \vee h(\widetilde{v_!}))/aw't'z = (a \wedge u')/aw't'z \vee h(\widetilde{v_!})/aw't'z = (a/aw't'z \wedge u'/aw't'z) \vee h(\widetilde{v_!})/aw't'z = (\top \wedge u'/w't'z) \vee h(\widetilde{v_!})/w't'z = u'/w't'z \vee h(\widetilde{v_!})/w't'z = (u' \vee h(\widetilde{e_!}))/w't'z$. By inductive hypothesis $(u' \vee h(\widetilde{v_!}))/w't'z = \top$.

The cases $\vee$ and $\cdot$ are similar. The proof can be generalized for any workflow preceding and following u' in u. □

It is worth noting that when the perturbation does not occur, we still have that $(u \vee h(\widetilde{v_!}))/wtz = \top$ because $u/wtz = \top$. In the above proposition, we have made the simplifying assumption that $h(\widetilde{v_!})$ is not pertubed by other perturbations. This assumption can be relaxed by introducing further recovery strategies, one for each perturbation affecting $h(\widetilde{v_!})$. This could be repeated indefinitely specifying recovery strategies for perturbations affecting recovery strategies. In practice the designer has to put a limit to the depth of this chain of perturbation handlers.

4.1 Accountability

We now explain how, by relying on Proposition 1, accountability can be used as an infrastructure for ensuring robustness against known perturbations. First of all, we need to characterize the accounts. Accountability, in fact, exists only when the agent providing an account is aware of the process it is accounting for, and has access to all the relevant information, possibly by gathering accounts from others. We thus say that the account is sound. To guarantee sound accounts, we define accountability on top of responsibility statements $\mathsf{R}(\cdot)$ by means of *accountability structure* (A-structure).

Definition 3 (A-structure). *Let $\mathbf{R}$ be a responsibility distribution, an* A-structure *over $\mathbf{R}$ is a pair $\langle A, T \rangle$, where:*

- *A is a set of accountability agreements $\{\mathbb{AA}(x_1, y_1, r_1, u_1), \ldots, \mathbb{AA}(x_n, y_n, r_n, u_n)\}$;*
- *T is either:*
  - *a set $\{\mathsf{R}(x_1, u_1), \ldots, \mathsf{R}(x_n, u_n)\} \subseteq \mathbf{R}$, such that for each accountability agreement $\mathbb{AA}(x_i, y_i, r_i, u_i) \in A$, there is a corresponding responsibility $\mathsf{R}(x_i, u_i) \in T$. This kind of A-structure is named* A-leaf*;*
  - *a set of A-structures.*

A-leaves represents one-step accounts: situations where the agent providing an account about a workflow is also responsible for the very same workflow, and this guarantees the provision of a sound account. Accountability characterizes A-structures so as to provide the same guarantee even when an agent accounts for workflows for which it is not directly responsible. To this aim, we define the following two operations upon A-structures.

Definition 4 (A-union). *The* A-union *between two A-structures, denoted by $\langle A_1, T_1\rangle \oplus \langle A_2, T_2\rangle$, is either:*

- *$\langle A_1 \cup A_2, T_1 \cup T_2\rangle$, if both the A-structures are A-leaves, or none of them is an A-leaf;*
- *$\langle A_1 \cup A_2, T_1 \cup \{\langle A_2, T_2\rangle\}\rangle$, if $\langle A_2, T_2\rangle$ is an A-leaf and $\langle A_1, T_1\rangle$ is not;*
- *$\langle A_1 \cup A_2, T_2 \cup \{\langle A_1, T_1\rangle\}\rangle$, if $\langle A_1, T_1\rangle$ is an A-leaf and $\langle A_2, T_2\rangle$ is not.*

Definition 5 (A-join). *The* A-join *between two A-structures, denoted by $\langle A_1, T_1\rangle \otimes \langle A_2, T_2\rangle$, is recursively defined as follows:*

- *$\langle A_1, T_1\rangle \oplus \langle A_2, T_2\rangle$, if $\langle A_1, T_1\rangle$ and $\langle A_2, T_2\rangle$ are A-leaves;*
- *$\langle A_1 \cup A_2, \{t_i \otimes t_j, \forall (t_i, t_j) \in T_1 \times T_2\}\rangle$, otherwise.*

We can define accountability as follows.

Definition 6 (Accountability). *Let u be a workflow and let $\mathbf{R}_u$ be a distribution of responsibility, an accountability $\mathbb{A}(x, y, r, u)$ over $\mathbf{R}_u$ is an A-structure $\langle\{\mathbb{AA}(x, y, r, u)\}, T\rangle$ defined as follow:*

- *T is $\{\mathsf{R}(x, u)\}$, such that $\mathsf{R}(x, u) \in \mathbf{R}_u$;*
- *$\mathbb{A}(x, y, r, u) = \mathbb{AA}(x, y, r, u) + \mathbb{A}(z, x, r, u)$;*
- *$\mathbb{A}(x, y, r, u' \vee u'') = \mathbb{A}(x, y, r, u') \vee \mathbb{A}(x, y, r, u'')$;*
- *$\mathbb{A}(x, y, r, u' \wedge u'') = \mathbb{A}(x, y, r, u') \wedge \mathbb{A}(x, y, r, u'')$;*
- *$\mathbb{A}(x, y, r, u' \cdot u'') = \mathbb{A}(x, y, r, u') \cdot \mathbb{A}(x, y, r \cdot u', u'')$.*

Where the operations $+$, $\vee$, $\wedge$, and $\cdot$ on accountabilities are defined as follows, supposing $\mathbb{A}(x, y, r, u') = \langle A_{u'}, T_{u'}\rangle$, $\mathbb{A}(x, y, r, u'') = \langle A_{u''}, T_{u''}\rangle$, $\mathbb{A}(z, x, r, u) = \langle A_u, T_u\rangle$:

- *$\mathbb{AA}(x, y, r, u) + \mathbb{A}(z, x, r, u) = \langle\{\mathbb{AA}(x, y, r, u)\}, \{\langle A_u, T_u\rangle\}\rangle$;*
- *$\mathbb{A}(x, y, r, u') \vee \mathbb{A}(x, y, r, u'') = \langle\{\mathbb{AA}(x, y, r, u' \vee u'')\}, \{\langle A_{u'}, T_{u'}\rangle \oplus \langle A_{u''}, T_{u''}\rangle\}\rangle$;*
- *$\mathbb{A}(x, y, r, u') \wedge \mathbb{A}(x, y, r, u'') = \langle\{\mathbb{AA}(x, y, r, u' \wedge u'')\}, \{\langle A_{u'}, T_{u'}\rangle \otimes \langle A_{u''}, T_{u''}\rangle\}\rangle$;*
- *$\mathbb{A}(x, y, r, u') \cdot \mathbb{A}(x, y, r \cdot u', u'') = \langle\{\mathbb{AA}(x, y, r, u' \cdot u'')\}, \{\langle A_{u'}, T_{u'}\rangle \otimes \langle A_{u''}, T_{u''}\rangle\}\rangle$.*

We use A_u as a shortcut for a singleton set $\{\mathbb{AA}(x, y, r, u)\}$ for some agents x and y, and some condition r and workflow u.

Example 2. In the yoghurt production example, let's consider responsibility distribution $\mathbf{R} = \{\mathsf{R}(\mathit{roboticArm}, \mathsf{takeCup}), \mathsf{R}(\mathit{fillingSensor}, \mathsf{enableFiller}), \mathsf{R}(\mathit{filler}, \mathsf{fillCup}), \mathsf{R}(\mathit{packager}, \mathsf{putLid})\}$. We define the accountability of the *supervisor* agent towards a *userAgent* about yoghurt production, over $\mathbf{R}$ as: $\mathbb{A}(\mathit{supervisor}, \mathit{userAgent}, \top, \mathsf{production})$. By Definition 6, this amounts to $a_1 \cdot a_2$ where $a_1 = \mathbb{A}(\mathit{supervisor}, \mathit{userAgent}, \top, (\mathsf{takeCup} \wedge \mathsf{enableFiller}) \cdot \mathsf{fillCup})$ and $a_2 = \mathbb{A}(\mathit{supervisor}, \mathit{userAgent}, (\mathsf{takeCup} \wedge \mathsf{enableFiller}) \cdot \mathsf{fillCup}, \mathsf{putLid})$. Again by Definition 6, a_1 can be decomposed as $a_3 \cdot a_4$, where $a_3 = \mathbb{A}(\mathit{supervisor}, \mathit{userAgent}, \top, \mathsf{takeCup} \wedge \mathsf{enableFiller})$, and $a_4 = \mathbb{A}(\mathit{supervisor}, \mathit{userAgent}, \mathsf{takeCup} \wedge \mathsf{enableFiller}, \mathsf{fillCup})$. Focusing on a_4, we have that $a_4 = \mathbb{AA}(\mathit{supervisor}, \mathit{userAgent}, \mathsf{takeCup} \wedge \mathsf{enableFiller}, \mathsf{fillCup}) + a_5$, where a_5 is $\mathbb{A}(\mathit{filler}, \mathit{supervisor}, \mathsf{takeCup} \wedge \mathsf{enableFiller}, \mathsf{fillCup})$. Here, the accountability agreement between *supervisor* and *userAgent* w.r.t. fillCup is supported by a_5, in which *supervisor* is a-taker and *filler* is a-giver. Notice also that a_5 is an A-leaf since $a_5 = \langle\{\mathbb{AA}(\mathit{filler}, \mathit{supervisor}, \mathsf{takeCup} \wedge \mathsf{enableFiller}, \mathsf{fillCup})\}, \{\mathsf{R}(\mathit{filler}, \mathsf{fillCup})\}\rangle$ thanks to the initial responsibility distribution $\mathbf{R}$. The structure of accountability ensures that *userAgent* will be in condition to receive an authoritative account about fillCup from the *supervisor*. Despite not being directly involved in the task, *supervisor* can recursively gather an account thanks to the *filler*'s accountability. The same construction can be applied to the steps in the production workflow.

Definition 6 says accountability is a tree by construction, whose nodes have form $\mathbb{A}(x, y, r, u) = \langle\{\mathbb{AA}(x, y, r, u)\}, T_u\rangle$. $\mathbb{AA}(x, y, r, u)$ is an accountability agreement, and T_u is the structure through which a sound account of u can be delivered to x. Grounding the tree over responsibility statements (A-leaves), assures that each agent, involved in the structure, has the means for gaining situational awareness of the process it is concerned with. So, it can provide sound accounts. We formalize this as a property of accountable workflows.

Definition 7 (Feedback chain). *A feedback chain is a sequence of $\langle\mathbb{A}(x_0, y_0, r_0, u_0), \ldots, \mathbb{A}(x_n, y_n, r_n, u_n)\rangle$ such that for each $i = 1, \ldots, n$, we have that $u_{i-1}[u_i]$, $y_i = x_{i-1}$, and $r_i \longrightarrow r_{i-1}$.*

Proposition 2 (Accountable workflow). *Let u be a workflow and let $\mathbb{A}(x, y, r, u)$ be an accountability over $\mathbf{R}_u$. There exists a feedback chain $\langle\mathbb{A}(x, y, r, u), \ldots, \mathbb{A}(x', y', r', u')\rangle$ for any workflow u' such that $u[u']$. We call u an accountable workflow through $\mathbb{A}(x, y, r, u)$ over $\mathbf{R}_u$.*

Proof. Let us denote as $ST(\mathbb{A}(x, y, r, u))$ the syntax tree of the workflow on $\mathbb{A}(x, y, r, u)$. The proof is by induction on the depth of the syntax tree $ST(\mathbb{A}(x, y, r, u))$.

In the case of depth 1, $\mathbb{A}(x, y, r, u)$ is $\langle\{\mathbb{AA}(x, y, r, u)\}, \{\mathsf{R}(x, u)\}\rangle$, where u is atomic, and the chain is given by $\langle\mathbb{A}(x, y, r, u)\rangle$.

If the depth is n. In case u is $u_1 \wedge u_2$, then $\mathbb{A}(x, y, r, u_1 \wedge u_2) = \mathbb{A}(x, y, r, u_1) \wedge \mathbb{A}(x, y, r, u_2)$. Since by hypothesis $u[u']$, then we have either $u_1[u']$ or $u_2[u']$. Let us assume $u_1[u']$. By inductive hypothesis, for any workflow u' such that $u_1[u']$ there exists $\langle\mathbb{A}(x, y, r, u_1), \ldots, \mathbb{A}(x', y', r', u')\rangle$. Now, if we substitute the first element of this chain with $\mathbb{A}(x, y, r, u)$, we get again feedback chain. In fact, by initial hypothesis $\mathbb{A}(x, y, r, u)$ is an A-framework, and by construction $u[u_1[u']]$ holds. The cases $\vee$ and $\cdot$ are similar.

Finally, in case $\mathbb{A}(x, y, r, u) = \mathbb{AA}(x, y, r, u) + \mathbb{A}(z, x, r, u)$, we have by inductive hypothesis, for any workflow u' such that $u[u']$ there exists $\langle\mathbb{A}(z, x, r, u), \ldots, \mathbb{A}(x', y', r', u')\rangle$. Thus, the sequence $\langle\mathbb{A}(x, y, r, u), \mathbb{A}(z, x, r, u), \ldots, \mathbb{A}(x', y', r', u')\rangle$ is again a feedback chain, and this proves the proposition. □

Proposition 2 guarantees that, given a workflow u, it is possible to define a proper set of feedback chains, from a-givers to the corresponding a-takers, for every sub-workflow of u. Thanks to responsibility, a-givers are the agents that are competent for each subprocess of u, and hence can provide sound accounts.

Example 3. With reference to Example 2, thanks to $\mathbb{A}(\textit{supervisor}, \textit{userAgent}, \top, \text{production})$ over $\mathbf{R}$, we can see that production is an accountable workflow. For each subworkflow, due to the responsibilities in $\mathbf{R}$ and to the structure imposed by the accountability, it is possible to find a suitable feedback chain. For instance, for fillCup, the feedback chain is: $\langle \mathbb{A}(\textit{supervisor}, \textit{userAgent}, \top, \text{production}), \mathbb{A}(\textit{filler}, \textit{supervisor}, \text{takeCup} \wedge \text{enableFiller}, \text{fillCup})\rangle$.

Accountability is preserved at runtime with respect to the events that occur during the execution.

Proposition 3 (Accoutability persistency). *Let u be a workflow and let $\mathbf{R}_u$ be a responsibility distribution. Given $\mathbb{A}(x, y, r, u)$ over $\mathbf{R}_u$ and an event e such that $r/e \neq 0$, we have that $\mathbb{A}(x, y, r/e, u)$ holds over $\mathbf{R}_u$.*

Proof. The proof is by induction on the structure of $\mathbb{A}(x, y, r, u)$. In the base case, $\mathbb{A}(x, y, r, u)$ is an A-leaf $\langle\{\mathbb{AA}(x, y, r, u)\}, \{\mathsf{R}(x, u)\}\rangle$. By definition of progression on $\mathbb{AA}(\cdot)$, it follows that $\mathbb{A}(x, y, r, u)/e = \langle\{\mathbb{AA}(x, y, r/e, u)\}, \{\mathsf{R}(x, u)\}\rangle = \mathbb{A}(x, y, r/e, u)$: the awareness of the agent x does not change after the progression of the context r under the occurrence of event e.

In the general case, we distinguish two situations. First, let $\mathbb{A}(x, y, r, u)$ be $\mathbb{A}(x, y, r, u')$ *op* $\mathbb{A}(x, y, r, u'')$ (where $op \in \{\vee, \wedge, \cdot\}$), and by inductive hypothesis let $\mathbb{A}(x, y, r/e, u')$ and $\mathbb{A}(x, y, r/e, u'')$ hold. It follows immediately that also $\mathbb{A}(x, y, r/e, u)$ holds. Second, let $\mathbb{A}(x, y, r, u)$ be $\mathbb{AA}(x, y, r, u) + \mathbb{A}(z, x, r, u)$, also in this case the thesis follows directly by the inductive hypothesis $\mathbb{AA}(x, y, r/e, u)$ and $\mathbb{A}(z, x, r/e, u)$. ◻

4.2 Robustness through Accountability

Proposition 3 assures that accountability is preserved against progression. Thus, this is exactly the structure upon which a robustness property can be put in effect. Intuitively, when a workflow is robust to a perturbation with account $\widetilde{v_!}$, not only there exists a specific recovery strategy $h(\widetilde{v_!}) = \widetilde{v_!} \cdot \overleftarrow{v_!}$, but, thanks to accountability, it is also guaranteed that the account of the perturbation will actually be available. By providing the account $\widetilde{v_!}$ we get two results: first, it is possible to notify the repairing agent that something wrong has occurred, and a repair is needed; second, it is possible to provide the repairing agent with information that help understand how to handle the situation. This is important when agents, as often happens, do not have a complete view of events. The illustration with JaCaMo, in the next section, exploits this feature.

Proposition 4 (Account Availability). *Let $u = u' \vee h(\widetilde{v_!})$ be a workflow such that $u'[v]$, and let $\widetilde{v_!}$ be the account for a perturbation on v. Let $\mathbf{R}_u$ be a distribution of responsibility, such that $\mathsf{R}(y, u)$ belongs to $\mathbf{R}_u$. If u' is an accountable workflow through $\mathbb{A}(x, y, r, u')$ over $\mathbf{R}_u$, then the account is available to the agent y in charge of the recovery strategy for the perturbation.*

Proof. The proof follows directly from propositions 1 and 2. From Proposition 1, we have that a workflow u' such that $u'[v]$ is robust to a perturbation with account $\widetilde{v_!}$ if a dedicated recovery strategy $h(\widetilde{v_!}) = \widetilde{v_!} \cdot \overleftarrow{v_!}$ is activated instead of u' when $\widetilde{v_!}$ occurs. On the other hand, the account $\widetilde{v_!}$ must be generated somewhere in the system. This is granted by Proposition 2: since u' is an accountable workflow, there is always the chance to generate an account for any of its sub-workflow, including v, even when it is perturbed. There exists, in fact, a feedback chain $\langle \mathbb{A}(x, y, r, u'), \ldots, \mathbb{A}(x', y', r', v)\rangle$, where the account $\widetilde{v_!}$ is first generated by x' and then propagated to y, responsible for the whole workflow $u = u' \vee h(\widetilde{v_!})$. ◻

Example 4. Let us consider a perturbation outOfStock, concerning putLid, and a corresponding recovery event orderProduct. We can extend the responsibility distribution $\mathbf{R}$ so as to make the production workflow robust by adding the responsibility $\mathsf{R}(\textit{supervisor}, \text{orderProduct})$ and turning the workflow into (production $\vee$ orderProduct) – *supervisor* will also be in charge of the treatment. Should a perturbation outOfStock occur, by way of the new responsibilities, an account would be provided by *filler* to *supervisor*; this, in turn, would exploit the account for activating a recovery strategy.

5 ILLUSTRATION IN JACAMO

JaCaMo [12] is a conceptual model and programming platform that integrates agents (programmed in Jason [13]), environments (programmed in CArtAgO [38]) and organizations (programmed in Moise [32]). A Moise organization has three dimensions. The structural dimension specifies roles, groups and links between roles in the organization. The functional dimension is made of schemes, which elicit how the global organizational goal is decomposed into subgoals, and how subgoals are grouped in coherent sets, called missions – to be distributed to the agents. The normative dimension binds the two previous dimensions by specifying permissions and obligations that are associated with each role. At the beginning of the execution the agents, playing the different roles, are asked to *commit* to certain missions, as specified by the norms. The organization will then issue obligations to achieve mission goals to the committed agents to coordinate the distributed execution.

On this basis, similarly to what done in JaCaMo+ for commitments [4], we map the concepts of `Mission` and `Goal` from JaCaMo onto the concepts of `Responsibility` and `Task` of the proposed conceptual model, Figure 1. We interpret the agent's commitment to a mission as an assumption of responsibility. Indeed, in JaCaMo agents commit to missions before pursuing (part of) the organizational goal and, from that moment on, the organization can issue obligations towards them to make them achieve the mission goals. To fulfill an obligation about a mission goal, an agent maps it into an internal goal, and tries to satisfy it. By accomplishing such an internal goal, the agent carries out the task the organization demanded by issuing the obligation. Hence, achieving that internal goal amounts to achieving the original mission goal.

In order to represent and interpret norms, JaCaMo uses the Normative Programming Language (NPL) [31] A norm in this language has the following syntax: `norm` *id* : φ `->` ψ, where *id* is an identifier of the norm, φ is the activation condition of the norm, and ψ is the consequence of the norm. A consequence can either be the issue of an obligation, or a failure. The former is used to raise obligations toward agents about goals to be achieved. The latter is used to model regimented norms; e.g., conditions that are not allowed. Intuitively, when ϕ is `fail`, any agent action that makes φ true will fail, too (and no change in the organization occurs).

We have seen that accountability $\mathbb{A}(x, y, r, u)$ is a pair $\langle\{\mathbb{AA}(x, y, r, u)\}, T_u\rangle$ where the $\mathbb{AA}(.)$ captures an agreement, and amounts to the normative dimension, while T_u captures the structure supporting the agreement. In order to introduce robustness through accountability inside JaCaMo, we now map the normative dimension of accountability within the normative system of the organization. For what concerns the structural dimension, this is a property that can be verified by assessing whether, given a distribution of responsibilities (i.e., a set of JaCaMo missions), it holds that for each obligation that x has about accounting on u, x has the means for generating an account either because it is directly responsible, or thanks to other accounts that x is legitimated to ask to other agents, and so forth recursively. As explained, responsibility is not directly represented in JaCaMo, but we can see the commitment to a mission as a declaration of responsibility assumption.

Normative dimension. We can realize the normative dimension of accountability by defining suitable norms in NPL. The obligation to provide an account, induced by an accountability agreement $\mathbb{AA}(x, y, r, u)$, is generated by the following norm.

```
1  norm nAccountProduction:
2      accountabilityAgreement(Request_u, Account_u, R) & R &
3      request(Request_u, Requesting_goal_u) &
4      account(Account_u, Accounting_goal_u) &
5      done(S, Requesting_goal_u, Y) &
6      mission_goal(M1, Requesting_goal_u) & committed(S,M1,Y) &
7      mission_goal(M2, Accounting_goal_u) & committed(S,M2,X)
8  -> obligation(X, enabled(S, Accounting_goal_u),
9       done(S, Accounting_goal_u, X), ...).
```

The norm specifies that, when y is legitimated to ask an account about u to x in r, an obligation towards x to achieve goal `Accounting_goal_u` is issued. `Accounting_goal_u` will make x provide y with the requested account. Legitimation is due to the fact that there exists a suitable accountability agreement for the current context r (Lines 2-4), y asks for the account (Line 5), y's requesting goal is part of its mission (Line 6), and x is competent for producing an authoritative account about u because this is part of its mission (Line 7). Agent y has the permission to ask for an account only when such a request is part of its mission, and condition r holds. In NPL this can be expressed by means of the following two norms: one that prohibits y to ask for a report when the context does not hold, and another that prohibits y to ask when the requesting goal does not belong to its mission.

```
norm nContextNotHolding:
    accountabilityAgreement(Request_u, Account_u, R) & not R &
    request(Request_u, Requesting_goal_u) &
    done(S, Requesting_goal_u, Y) &
    mission_goal(M, Requesting_goal_u) & committed(S,M,Y))
-> fail(contextNotHolding(Y, Request_u, R)).
```

The argument of the `fail` operator, `contextNotHolding(Y, Request_u,R)`, represents the reason for the failure.

```
norm nRequestNotAllowed:
    accountabilityAgreement(Request_u, Account_u, R) &
    request(Request_u, Requesting_goal_u) &
    done(S, Requesting_goal_u, Y) &
    not (mission_goal(M, Requesting_goal_u) & committed(S,M,Y))
-> fail(notLegitimateRequest(Y, Request_u)).
```

Robustness. In order to gain robustness, it is necessary that when an agent x detects a perturbation about a workflow v it is responsible for, x send an account $\widetilde{v_!}$, about such a perturbation, to the agent mandated the execution of a recovery strategy. Assuming to have the accountability $\mathbb{A}(x, y, r, v)$, by Proposition 4 we introduce the responsibilities $\mathsf{R}(x, v)$ and $\mathsf{R}(y, u \vee h(v_!))$, where $u[v]$. The former responsibility just denotes that x is in charge of doing v. (Note that agent x is also account giver towards y about v.) The latter responsibility is attributed to y, and denotes that y is in charge of workflow u including the recovery strategy handling the account $\widetilde{v_!}$ in case a perturbation with such an account should affect v. To achieve this result in JaCaMo it is sufficient to define two missions, one for agent x and one for agent y, mentioning respectively, goal v and goal $u \vee h(v_!)$, so that $u[v]$. And then, concretize $\mathbb{A}(x, y, r, v)$ via a proper set of norms as explained above. The last step toward robustness is to push y to activate the recovery strategy when it receives the corresponding account. This can be obtained by the following norm.

```
1  norm nTreatAccount:
2      accountabilityAgreement(Request_u, Account_u, R) &
3      request(Request_u, Requesting_goal_u) &
4      account(Account_u, Accounting_goal_u) &
5      recoveryStrategy(Account_u, Treatment_goal_u) &
6      done(S, Requesting_goal_u, Y) & done(S, Accounting_goal_u, X) &
7      mission_goal(M1, Requesting_goal_u) & committed(S,M1,Y)) &
8      mission_goal(M2, Accounting_goal_u) & committed(S,M2,X)) &
9      mission_goal(M3, Treatment_goal_u)  & committed(S,M3,Y))
10 -> obligation(X, enabled(S, Treatment_goal_u),
11                 done(S, Treatment_goal_u, Y), ...).
```

In words, when a requested account is delivered to the a-taker, and a recovery strategy is applicable to such an account (see Line 5), the a-taker, by virtue of its responsibility regarding the associated treatment task (Line 9), becomes obliged to activate that treatment task. Such an obligation does not specify what the agent is actually asked to do to tackle the perturbation. The treatment task is, in fact, mapped into a local agent goal, as usual. Autonomy is, then, preserved and the agent will be free to apply the recovery strategy which suits best the provided account and its expertise.

It is worth noting that, although accountability is a directed relationship between agents, it is realized in JaCaMo by means of undirected obligations. This happens because in order to realize accountability, we rely on the organization's normative system, by way of concepts like obligations and goals. In other approaches, such as [6, 18] where agents establish their accountability agreements by way of protocols, it would be possible gain robustness by relying on social commitments, that differently from obligations, are always directed from a debtor towards a creditor.

6 RELATED WORKS AND CONCLUSION

Building upon suggestions from many works, we have defined a constructive technical framework of accountability (Definition 6) for supporting the realization of robust multiagent organizations, and we have illustrated how the framework can be mapped onto the JaCaMo agent platform. An accountability relationship grants legitimacy to the a-taker to ask for an account, and to the a-giver to provide a meaningful account. The legitimacy of the a-giver is grounded on the *structural* dimension of accountability, that makes it a reliable source about events of interest (e.g., perturbations).

This is the distinctive feature of accountability w.r.t. other social relationships like business contracts and social commitments, which only yield obligations to do something (*normative* dimension). The proposal features a clear separation of concerns: accountability agreements anticipate at design time the kinds of perturbation of interest; how these are actually handled, however, depends on the specific plans (or behaviors) implemented by the agents, playing organizational roles (which are known only at runtime). Second, our mechanism is useful to handle perturbations that cannot be properly addressed by a single agent. In many cases, the agent that detects a perturbation is not aware of the global context (e.g., how the perturbation may indirectly affect tasks of other agents), and has no power for fixing the problem. On the other hand, the agent that could handle the perturbation has no access to the situation where the perturbation has occurred. Accountability is the means through which an account about a perturbation is reported to the agent who is responsible for treating that perturbation. In some way, accountability complements the plan failure mechanism of JaCaMo (where failure is tackled by an agent locally) because it enables a sort of escalation of a failure.

This work sets the ground for several future directions. First, it represents a general schema that can be tailored to specific applications, e.g., to realize an exception handling mechanism in agent organizations, by constraining the way in which agents produce and consume accounts. The current proposal builds robustness on top of a given distribution of responsibility but we mean to extend the study to the cases where responsibilities may change along time, either by effect of the received accounts or due to external circumstances; e.g., an agent leaving the organization or a bottleneck that is identified and solved by splitting some responsibilities among many agents. Another future direction of work concerns the realisation of tools that can support the work of actual organizations, like the mentioned UNDP, by combining accountability frameworks with oversight policies. This case is more general than the one we have tackled because accounts will often concern achievements, and will also allow taking advantage of opportunities.

In software engineering, robustness is considered a key property of software systems [33], and is usually gained by ensuring (at design time) that "exceptional" events will be reported to those software components which have the means for handling them properly. As pinned out in [37], traditional exception handling approaches, however, do not fit some key characteristics of multiagent systems, like openness, heterogeneity, agent encapsulation, and distribution. In particular, they usually assume that software components are collaborative, and that their code can be inspected while handling some given exception. But introspection is often impossible when dealing with agents, and collaboration cannot be given for granted. [37] suggests that an exception handling mechanism for multiagent systems should leverage both on the proactivity of agents, and on the environment in which agents are situated. Nevertheless, a few authors faced the problem of modeling exceptions in an agent-based system. Among them, [35] relies on commitment-based protocols, while [29] proposes an obligation-based approach for exception handling in interaction protocols.

In [15, 16] the authors explain how, within Socio-Technical Systems, accountability plays a fundamental role in balancing the principals' autonomy: a principal can decide to violate any expectation for which it is accountable, however, by way of accountability the principal would be held to account about that violation.

Accountability is recognized as a value for developing software also in [19], where a proposal complementary to ours is made. There, the authors focus on answer production in presence of an accountability relationship, tacking questions: how to properly define the temporal window to consider? Which pieces of information are relevant in this time interval? Which questions are suitable to be asked in this setting? The account giving agent produces an answer in terms of its internal mechanisms. The proposal, however, does not provide the organizational view of the system of interacting agents and does not tackle robustness and exceptions.

In [17], accountability enables the process of norms adaptation by feeding outcomes back into the design-phase. In this approach, the account is a justification of an agent's norm-violating behavior. This is a different understanding of accounts than ours because, for us, account givers are not rule violators: they meet perturbations, and provide information about the encountered situations. The account takers, on their hand, will interpret the received accounts – possibly combining them with further information provided by other agents or that simply belongs to the callee's level. The adaptation process in [17], that consists in norm modification, however, can be seen as a kind of robustness. Our objective is different: we do not target norm modification, but the achievement of the organizational goal despite the occurrence of perturbations. The two approaches are not in contrast, rather, they complement each other. They are both exemplifications of the perspective put forward in [1], for which a property of a system is robust if it is invariant with respect to a set of perturbations. The difference lies in the type of perturbations the two approaches aim at.

Finally, MOCA [7] provides an information model of accountability, that captures the kind of facts that must be available to allow the identification of account-givers in certain situation of interest. The model is given in Object-Role Modeling (ORM) [30] due to the relational nature of the represented concepts, and enables automatic verification of consistency The information model is centered around two basic concepts: *just expectation* and *control*. Just expectation is intended as the mutual awareness and acceptance of an accountability relationships between the involved a-giver and a-taker. Control, instead, is intended as the power, possibly exerted indirectly by means of other agents, of achieving a condition of interest. The normative and structural dimensions of accountability, that characterize our proposal, respectively capture these two features. Through the normative dimension, agents are aware of the obligations they may be subjected as a-givers, and what permissions they have as a-takers. The structural dimension, instead, grounds accountability relationships over an explicit assumption of responsibility by the agents, that we interpret as a declaration of direct control.

ACKNOWLEDGMENTS

We thank the reviewers for the constructive comments, prof. Olivier Boissier for the discussions. Stefano Tedeschi is partially supported by Progetto Talenti Società Civile Fondazione CRT, Fondazione G. Goria.

REFERENCES

[1] David L. Alderson and John C. Doyle. 2010. Contrasting Views of Complexity and Their Implications For Network-Centric Infrastructures. *IEEE Trans. Systems, Man, and Cybernetics, Part A* 40, 4 (2010), 839–852. https://doi.org/10.1109/TSMCA.2010.2048027

[2] Huib Aldewereld, Olivier Boissier, Virginia Dignum, Pablo Noriega, and Julian Padget (Eds.). 2016. *Social Coordination Frameworks for Social Technical Systems*. Law, Governance and Technology Series, Vol. 30. Springer International Publishing. https://doi.org/10.1007/978-3-319-33570-4

[3] Matteo Baldoni, Cristina Baroglio, Olivier Boissier, Katherine M. May, Roberto Micalizio, and Stefano Tedeschi. 2018. Accountability and Responsibility in Agents Organizations. In *PRIMA 2018: Principles and Practice of Multi-Agent Systems, 21st International Conference (Lecture Notes in Computer Science, 11224)*. Springer, Tokyo, Japan, 403–419.

[4] Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. 2018. Commitment-based Agent Interaction in JaCaMo+. *Fundamenta Informaticae* 159, 1-2 (2018), 1–33.

[5] Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. 2018. Type Checking for Protocol Role Enactments via Commitments. *Journal of Autonomous Agents and Multi-Agent Systems* 32, 3 (May 2018), 349–386.

[6] Matteo Baldoni, Cristina Baroglio, Katherine M. May, Roberto Micalizio, and Stefano Tedeschi. 2018. Computational Accountability in MAS Organizations with ADOPT. *Applied Sciences* 8, 4 (2018).

[7] Matteo Baldoni, Cristina Baroglio, Katherine M. May, Roberto Micalizio, and Stefano Tedeschi. 2019. MOCA: An ORM MOdel for Computational Accountability. *Journal of Intelligenza Artificiale* 13, 1 (2019), 5–20. https://doi.org/10.3233/IA-180014

[8] Matteo Baldoni, Cristina Baroglio, and Roberto Micalizio. 2020. Fragility and Robustness in Multiagent Systems. In *Post-Proc. of the 8th International Workshop on Engineering Multi-Agent Systems, EMAS 2020, Revised Selected Papers (LNAI, 12589)*, C. Baroglio, J. F. Hubner, and M. Winikoff (Eds.). Springer, Auckland, New Zealand, 61–77. https://doi.org/10.1007/978-3-030-66534-0

[9] Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, and Stefano Tedeschi. 2020. Is Explanation the Real Key Factor for Innovation?. In *Proceedings of the Italian Workshop on Explainable Artificial Intelligence co-located with 19th International Conference of the Italian Association for Artificial Intelligence, XAI.it@AIxIA 2020, Online Event, November 25-26, 2020 (CEUR Workshop Proceedings, Vol. 2742)*, Cataldo Musto, Daniele Magazzeni, Salvatore Ruggieri, and Giovanni Semeraro (Eds.). CEUR-WS.org, 87–95.

[10] B. Bauer, J.P. Müller, and J. Odell. 2001. Agent UML: A formalism for specifying multiagent software systems. *Software Engineering and Knowledge Engineering* 11, 3 (2001), 207–230.

[11] Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen. 2007. Introduction to Normative Multiagent Systems. In *Normative Multi-agent Systems (Dagstuhl Seminar Proceedings, Vol. 07122)*.

[12] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. 2013. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming* 78, 6 (2013), 747 – 761. https://doi.org/10.1016/j.scico.2011.10.004

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

[14] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. 2004. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems* 8, 3 (2004), 203–236. https://doi.org/10.1023/B:AGNT.0000018806.20944.ef

[15] Amit K. Chopra and Munindar P. Singh. 2014. The thing itself speaks: Accountability as a foundation for requirements in sociotechnical systems. In *IEEE 7th Int. Workshop RELAW*. IEEE Computer Society. https://doi.org/10.1109/RELAW.2014.6893477

[16] Amit K. Chopra and Munindar P. Singh. 2016. From social machines to social protocols: Software engineering foundations for sociotechnical systems. In *Proceedings of the 25th International Conference on World Wide Web*. 903–914.

[17] Amit K. Chopra and Munindar P. Singh. 2018. Sociotechnical Systems and Ethics in the Large. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, AIES 2018, New Orleans, LA, USA, February 02-03, 2018*, Jason Furman, Gary E. Marchant, Huw Price, and Francesca Rossi (Eds.). ACM, 48–53.

[18] Amit K. Chopra and Munindar P. Singh. 2020. Clouseau: Generating Communication Protocols from Commitments. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*. AAAI Press, 7244–7252.

[19] Stephen Cranefield, Nir Oren, and Wamberto Weber Vasconcelos. 2018. Accountability for Practical Reasoning Agents. In *Agreement Technologies - 6th International Conference, AT 2018, Bergen, Norway, December 6-7, 2018, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11327)*, Marin Lujak (Ed.). Springer, 33–48. https://doi.org/10.1007/978-3-030-17294-7_3

[20] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. 1993. Goal-directed requirements acquisition. *Science of Computer Programming* 20, 1 (1993), 3 – 50. https://doi.org/10.1016/0167-6423(93)90021-G

[21] Maiquel de Brito, Jomi Fred Hübner, and Olivier Boissier. 2017. Situated artificial institutions: stability, consistency, and flexibility in the regulation of agent societies. *Autonomous Agents and Multi-Agent Systems* (2017), 1–33. https://doi.org/10.1007/s10458-017-9379-3

[22] Virginia Dignum, Frank Dignum, and John-Jules Meyer. 2004. An agent-mediated approach to the support of knowledge sharing in organizations. *The Knowledge Engineering Review* 19, 2 (2004), 147–174.

[23] Virginia Dignum, Javier Vázquez-Salceda, and Frank Dignum. 2004. OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organizations. In *Programming Multi-Agent Systems, Second International Workshop ProMAS, Selected Revised and Invited Papers (Lecture Notes in Computer Science, Vol. 3346)*. Springer, 181–198.

[24] Melvin J. Dubnick and Jonathan B. Justice. 2004. Accounting for Accountability. https://pdfs.semanticscholar.org/b204/36ed2c186568612f99cb8383711c554e7c70.pdf Annual Meeting of the American Political Science Association.

[25] Dave Elder-Vass. 2011. *The Causal Power of Social Structures: Emergence, Structure and Agency*. Cambridge University Press.

[26] Executive Board of the United Nations Development Programme and of the United Nations Population Fund. 2008. *The UNDP accountability system, Accountability framework and oversight policy*. Technical Report DP/2008/16/Rev.1. United Nations.

[27] Christophe Feltus. 2014. *Aligning Access Rights to Governance Needs with the Responsability MetaModel (ReMMo) in the Frame of Enterprise Architecture*. Ph.D. Dissertation. University of Namur, Belgium.

[28] Harold Garfinkel. 1967. *Studies in ethnomethodology*. Prentice-Hall Inc., Englewood Cliffs, New Jersey.

[29] J. Octavio Gutierrez-Garcia, Jean-Luc Koning, and Félix F. Ramos-Corchado. 2009. An Obligation Approach for Exception Handling in Interaction Protocols. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Vol. 3. 497–500.

[30] Terry Halpin and Tony Morgan. 2008. *Information Modeling and Relational Databases*. Morgan Kaufmann Publishers.

[31] Jomi F. Hübner, Olivier Boissier, and Rafael H. Bordini. 2011. A normative programming language for multi-agent organisations. *An. of Math. and Art. Intel.* 62, 1 (2011), 27–53.

[32] Jomi F. Hubner, Jaime S. Sichman, and Olivier Boissier. 2007. Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels. *Int. J. Agent-Oriented Softw. Eng.* 1, 3/4 (2007), 370–395. https://doi.org/10.1504/IJAOSE.2007.016266

[33] ISO/IEC/IEEE. 2017. Systems and software engineering — Vocabulary.

[34] Fabiola López y López and Michael Luck. 2003. Modelling Norms for Autonomous Agents. In *4th Mexican International Conference on Computer Science (ENC 2003), 8-12 September 2003, Apizaco, Mexico*. IEEE Computer Society, 238–245. https://doi.org/10.1109/ENC.2003.1232900

[35] Ashok U. Mallya and Munindar P. Singh. 2005. Modeling Exceptions via Commitment Protocols. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '05)*. ACM, New York, NY, USA, 122–129.

[36] Roberto Micalizio and Pietro Torasso. 2014. Cooperative Monitoring to Diagnose Multiagent Plans. *J. Artif. Intell. Res.* 51 (2014), 1–70. https://doi.org/10.1613/jair.4339

[37] Eric Platon, Nicolas Sabouret, and Shinichi Honiden. 2007. Challenges for Exception Handling in Multi-Agent Systems. In *Software Engineering for Multi-Agent Systems V*. Springer, Berlin, Heidelberg, 41–56.

[38] Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. 2009. *Environment Programming in CArtAgO*. Springer US, Boston, MA, 259–288.

[39] Barbara S. Romzek and Melvin J. Dubnick. 1987. Accountability in the Public Sector: Lessons from the Challenger Tragedy. *Public Administration Review* 47, 3 (1987).

[40] Munindar P. Singh. 1999. An Ontology for Commitments in Multiagent Systems. *Artif. Intell. Law* 7, 1 (1999), 97–113.

[41] Munindar P. Singh. 2003. Distributed Enactment of Multiagent Workflows: Temporal Logic for Web Service Composition. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*. ACM, 907–914.

[42] Munindar P. Singh. 2013. Norms as a basis for governing sociotechnical systems. *ACM TIST* 5, 1 (2013), 21. https://doi.org/10.1145/2542182.2542203

[43] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. 2003. Developing multiagent systems: The Gaia methodology. *ACM Trans. Softw. Eng. Methodol.* 12, 3 (2003), 317–370. https://doi.org/10.1145/958961.958963