Reimagining Robust Distributed Systems through Accountable MAS

Department: Head
Editor: Name, xxxx@email

Reimagining Robust Distributed Systems through Accountable MAS

Matteo Baldoni, Cristina Baroglio, Roberto Micalizio, Stefano Tedeschi
Università di Torino, Dipartimento di Informatica, Torino, Italy

***Abstract*—Robustness is an important property of software systems; the availability of proper feedback is crucial to obtain it, especially when a system consists of distributed and interconnected components. Multiagent Systems (MAS) are valuable for conceptualizing and implementing distributed systems, but they fall short in addressing robustness in a systematic way at design time. In this paper we provide a definition of accountability and show its use for designing robust MAS.**

THE INTRODUCTION

Distributed systems are often characterized by complex networks of functional dependencies defined among a heterogeneous set of actors, all interacting with each other to achieve goals, that otherwise they would not achieve, or not so easily. A challenge in realizing such a kind of systems is to guarantee *robustness*.

According to ISO/IEC/IEEE 24765 vocabulary, robustness is the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions – that we generally call *perturbations*. For dealing with perturbations, the components of a robust system must adapt their behavior to unexpected contextual conditions, showing autonomy in their decision process. Multiagent Systems (MAS) [1] are a particularly appropriate approach: MAS, indeed, offer powerful abstractions for realizing (complex) systems, that are made of cooperating autonomous parts (agents) that amount to loci of decision. However, approaches to MAS design and development fall short in addressing robustness in a systematic way. Namely, they do not devise specific mechanisms for alerting the right agents in the system about the occurrence of perturbations, and for providing information that enables the enactment of system-wise behaviors for tackling them. Indeed, the lack of mechanisms for producing, propagating, and processing information about perturbations makes a system fragile [2], [3].

In order to fill this gap, we take inspiration from the way in which many human organizations tackle similar problems by resorting on accountability frameworks (e.g., see UN TR DP/2008/16/Rev.1). The approach is meeting greater and greater success especially in non-

 Published by the IEEE Computer Society

government organizations, that work in contexts where partial achievement of the desired results is common, and perturbations occur frequently. Accountability frameworks answer to the need of gathering information, to analyse it, and then take appropriate measures, based on actual data. So, the organization may enact alternative operational procedures, or even modify its own goals, practices or structure. In this perspective, accountability, which in sociology is seen as a basic mechanism that allows individuals to constitute societies [4], becomes an instrument of the administrative power, through which organizations can ensure the compliance of their processes to predefined standards, and it also enables changes aimed at improving the organization [5], [6].

Thus, we rely on *accountability* as a key to design and develop robust distributed systems and organizations. The main contribution of this paper is a conceptual model that defines and formalizes accountability in multi-agent organizational contexts. The conceptual model provides guidance for the development of systems able to account about their internal processes. Generally, accounts foster flexibility, that is, the reactivity, proactivity, and social ability of the agents. The agents will react to accounts, act so as to obtain an account, interact with others while dealing with accounts. We specifically show the particular use of gaining robustness to perturbations by showing through examples an implementation of the model that extends the JaCaMo agent platform [8].

Accountability and MAS Robustness

Accountability is often associated with blame [9], either *post factum* (who is to blame for an act or an error that has occurred), or *pre factum* (who is blameworthy for errors not yet occurred), but this is a *partial view* that disregards the potential involved in relationships, concerning the ability and the designation to provide response about something to someone who is legitimated to ask. Indeed, in political sciences [10], accountability expresses a general recognition of the legitimacy of the authority of the parties that are involved in the relationship: one to exercise particular powers and the other to hold them to provide an account. By leveraging such characteristics, it is possible to devise systems that can, in a way, reconfigure when they meet some perturbations. To this aim: 1) relevant information should be asked to informed agents, and delivered by these in the right format, 2) agents should be supplied with the means for understanding who is entitled to ask what to whom, and under which circumstances, 3) an appropriate treatment, based on the available information, must be applied. In absence of such instruments, the agents which were unable to carry out their assigned tasks, and which detected some perturbation, will be considered as violators and receive sanctions, but no action will be taken to understand the situation and tackle it.

A Conceptual Model

Figure 1 shows our proposal for a conceptual model of organizational accountability. The basic idea is that the agents, for taking part to the organization, must be aware that, under certain conditions, they might be called to *answer* to requests by providing an account. This will happen because by entering in the organization they will be in position to produce the requested information and they will be expected to act accordingly. We use UML class diagrams, and express the model in terms of conceptual classes (rectangles) and associations (lines). Conceptual classes express the type of the involved objects; associations represent relations among the conceptual objects that belong to the conceptual classes.

Broadly speaking, an organization can be seen as a distribution of responsibilities [11]. Norms yield obligations, prohibitions, and authorizations about tasks, that agents are expected to fulfill. Norms are used to describe the ideal behavior of the agents in terms of their responsibilities and rights. Thereby, we say that Tasks are distributed among Agents by way of Responsibility assumptions: an agent is part of the organization only when it explicitly takes on the responsibility that concerns some task. Tasks can be complex and involve sub-tasks, consequently, the Responsibility for the overall task may be assigned to some agent while that of the sub-tasks to other agents. For instance, this may be useful when one agent should supervise a process, that involves many tasks, each performed by a specialized agent. A Responsibility is always associated with a single (possibly complex) Task, while a Task may not be associated with any Responsibility or, on the contrary, it may be associated with more

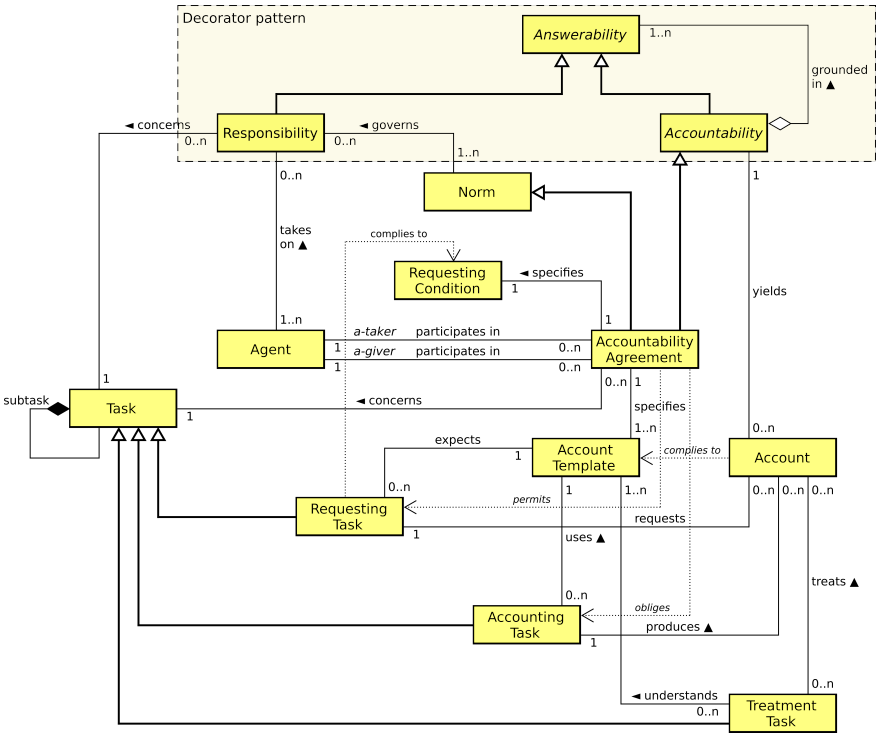

Figure 1. Organizational accountability: a conceptual model.

than one. Responsibilities are governed by Norms [11], that orchestrate the task execution according to the organizational goals. Indeed, it is not sufficient that an agent takes on the responsibility for a task. By generating obligations, permissions, etc., Norms signal to the agents when specific tasks should be performed.

Accountability and Answerability are abstract classes. Accountability grounds in Answerability (the ability to provide an answer), and is realized through a mutual understanding, captured by Accountability Agreement, by which the parts recognize each other's power (of producing a request and of producing an account, respectively [10]), together with the right of producing a request, under the stipulated conditions, and the obligation to produce an account as an answer, by following the modalities that are specified by the Accountability Agreement itself. In the literature [12], the party who is legitimately required to provide the account is commonly called "account giver", or *a-giver*, while the party who can legitimately ask, under some agreed conditions, to the other party an account about a process of interest is called the "account taker", or *a-taker*.

Accountability complements Responsibility in getting the overall organization answerable. This is captured by the *decorator pattern* (dashed box in Figure 1), involving Answerability, Accountability, Responsibility, and Accountability Agreement where Answerability is considered as an abstract concept modeling a desired property of the system as a whole. The pattern highlights two important features. First, accountability has a recursive structure. This enables an agent to provide accounts even for tasks that have been carried out by others, or that involve the intervention of others. The rationale is that an agent can provide an account about a (complex) task by gathering the accounts it receives from other agents concerning other (related) tasks. Second, the recursion always closes on a Responsibility. This implies that only tasks under the responsibility of some agents can be accounted for. More importantly, the recursive structure of accountability implies that an account is a reliable piece of information because it can always be traced back to the agents responsible for that

task. This explains why Accountability cannot be reduced to Accountability Agreement, that is, to the right/obligation involved in the relationship. Rather, we see accountability as having two main dimensions: *normative dimension*, capturing the legitimacy of asking and the availability to provide accounts, yielding expectations on the agents' behavior; *structural dimension*, capturing that, for being accountable about a task, an agent must have control over that task and awareness of the situation it will account for. The structural dimension is related to having power over a situation of interest [7], [13]. The model captures the *normative* dimension via Accountability Agreement, and the *structural* dimension, via the composite pattern. The Accountability Agreement between two parties specifies the a-taker and a-giver agents, the condition under which some request is allowed (Requesting Condition) and the structure of the account (Account Template, possibly many). By means of the association *concerns*, it establishes the task about which requests and accounts refer to. Account Templates are crucial because they allow a-taker and a-giver to tune by specifying the kind of Account one needs and expects from the other. In other words, a Requesting Task expects some Account Template to be followed. Conversely, the Account produced by an Accounting Task should comply to some Account Template. The characteristic of Requesting Tasks and Accounting Tasks, with respect to plain Tasks, is their relationship with an Account Template and with Account.

The structural dimension of accountability concerns the actual synthesis of an Account that relies on the explained recursive structure. Specifically, the a-taker must be an agent which takes responsibility for the given Requesting Task. The same holds for the a-giver and the Accounting Task. From a normative perspective, the a-taker is permitted to perform a Requesting Task in order to ask for an account. The a-giver may be obliged to perform an Accounting Task to produce an account. This is captured by means of the twofold association between Accountability Agreement and Agent. In one case, an agent plays the role of *a-taker*; in the other, of *a-giver*.

Tackling Perturbations

In distributed settings, that encompass interacting agents, action and decision depend on the availability of the right contextual information, that is, of the relevant causal dependencies between events. This can only be built in the right context. The problem is that such a context rarely is the one in which a perturbation occurs, especially in complex systems, where each agent has only a partial view of the overall ongoing process. Accountability allows integrating the management of perturbations in a seamless way. Indeed, norms govern the responsibilities about tasks that agents take on when taking part to an organization. Among these, there are those responsibilities which concern Requesting Tasks, and Accounting Tasks. Overall, in presence of perturbations (e.g., failures or some symptomatic situations that call for attention), it will be possible to ask and obtain accounts. This provides the ground for applying an adequate treatment, possibly revising the system behavior as done in human organizations. Actually, according to [14], accounts are expected to be used – typically by the account taker. In the model, accounts are treated by Treatment Tasks, and agents take on responsibility concerning Treatment Tasks. Such tasks can be complex and involve many agents, and it is also possible that an account is treated by way of many Treatment Tasks. Notice that Treatment Task is not part of Accountability Agreement, but rather it depends on the kind of robustness one wants to obtain, exploiting the structural and the normative dimensions of accountability.

Some Examples of Perturbations and Treatments Implemented

As an exemplification, we enriched JaCaMo [8] with the conceptual model in Figure 1 – http://di.unito.it/moiseaccountability. JaCaMo is a programming platform that integrates agents, environments and organizations. Moise, one of the elements of JaCaMo, implements the organization layer. It specifies roles and groups, a set of schemes that captures how the organizational goals are decomposed into sub-goals, grouped into missions, and a set of norms. Role-playing agents must take on the responsibility for mission goals. Triggered by the current system state,

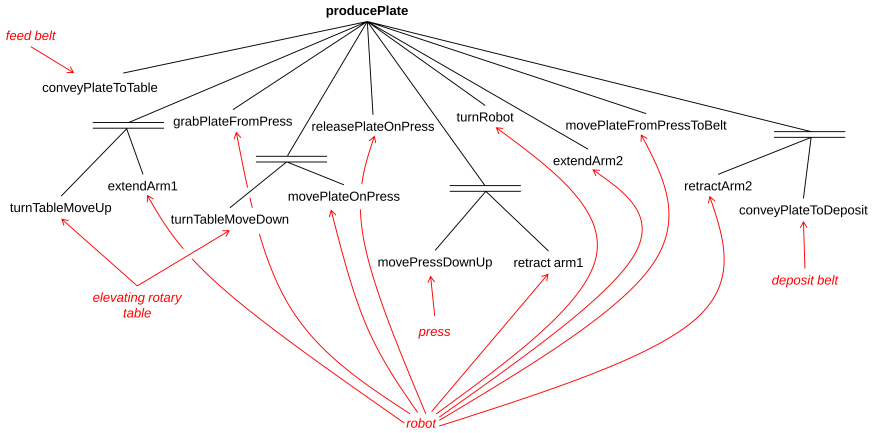

Figure 2. JaCaMo scheme of the industrial production cell.

norms issue obligations to achieve mission goals directed to the agents. Norms are thus used for coordinating the distributed execution. To fulfill an obligation, an agent maps it on one of its internal goals: the satisfaction of the latter will, then, amount to the achievement of the corresponding mission goal. This approach guarantees a strong decoupling between the agents and the organization, allowing agents to autonomously determine how to accomplish the organizational goals. In short, in JaCaMo, an organization is seen as a shared environment, where the satisfaction (or violation) of a responsibility makes the institutional situation evolve. Norms issue obligations accordingly. Agent programming amounts to specifying plans for satisfying incoming obligations while playing some role, so as to discharge all responsibilities.

Production Cell

The application scenario is inspired to the well-known production cell of the KorSo project, University of Karlsruhe 1989, which is widely used as a benchmark. It is made of five robots (agents): a feed belt, a depot belt, an elevating rotary table, a press, and a rotary robot with two extensible arms. Each device has sensors and actuators. The cell performs the following steps: 1) the feed belt conveys the plate from the storage rack to the elevating rotary table, 2) the table rotates and lifts, 3) the rotary robot takes the plate, 4) the rotary robot turns and places the plate onto the press, 5) the press forges the plate while the robot turns again, 6) the rotary robot picks up the forged plate and places it on the depot belt, and 7) the depot belt carries the plate forward to the depot. Figure 2 shows a JaCaMo scheme for the production of a metal plate – to be read from left to right. Some goals are to be achieved in sequence, while others (grouped under a double horizontal line) are pursued in parallel. In red the agents that are responsible for some organizational goal. We now detail how to tackle three perturbations within the schema.

Shortage of resources

Suppose there is a delay in the delivery of metal plates. Having the production cell come to a complete stop is expensive and to be avoided. The issue is handled by slowing down the production. Let's see the accountability agreement and treatment policy for this case. Note that JaCaMo uses XML to specify the organizational elements, that are turned into a body of norms. We show here a simplified version, highlighting the information that must be specified; the complete XML code is found at http://di.unito.it/moiseaccountability.

```
Account Template id: at1
  concerns: conveyPlateToTable
  requesting goal: requestRemainingStock
        condition: true
  accounting goal: notifyRemainingStock
         argument: availablePlates
  treatment goal: slowDownProduction
      condition: availablePlates <= 10
```

The Account Template specifies, via **concerns**, the goal (we use *goal* to align with JaCaMo's terminology, goals amount to Tasks)

about which an account can be asked (in the example `conveyPlateToTable`). The account can be asked, when its corresponding condition holds, by achieving the **requesting goal** `requestRemainingStock`. The account is produced through the achievement of the **accounting goal** `notifyRemainingStock`, by conveying the number of remaining plates `availablePlates`. **argument**, not limited to one occurrence, specifies the format of the account. Finally, the **treatment goal** `slowDownProduction` denotes how to handle this kind of perturbation. It is associated with a condition that specifies when the goal can be pursued; so, many treatment goals can be defined as alternative answers for the same perturbation. For instance the goal is activated when the number of plates is less than 10 pieces. Conditions can also mention the state of other organizational goals. This can be exploited to devise treatment goals that handle multiple perturbations.

An agent that takes responsibility for a requesting goal will be a-taker; an agent that takes responsibility for an accounting goal will be a-giver. So, the rotary robot assumes responsibility of `requestRemainingStock`, and hence is a-taker. The feed belt robot assumes responsibility of `notifyRemainingStock` and hence is a-giver. Next comes an excerpt of a Jason implementation of the rotary robot.

Jason agents have a BDI architecture including a belief base and a plan library storing the set of plans available to the agent for execution. A Jason program is a set of plans of kind $event : context \leftarrow body$, where $event$ denotes the event the plan handles (e.g. adding "+" a belief or a goal), $context$ is the circumstance when the plan can be used, and $body$ says what should be done. Operator "!" achievement goals.

```
+delay[source(supplier)]
    <- !requestRemainingStock.

+!requestRemainingStock
    <- // Perform the request...
       goalAchieved(requestRemainingStock).

+obligation(Ag,_,done(_,slowDownProduction,Ag),_)
     : .my_name(Ag) & availablePlates(N) & N >= 5
    <- setProductionSpeed(0.7); // set speed to 70%
       goalAchieved(slowDownProduction).

+obligation(Ag,_,done(_,slowDownProduction,Ag),_)
     : .my_name(Ag) &
       availablePlates(N) & N < 5 & N >= 2
    <- setProductionSpeed(0.3);
       goalAchieved(slowDownProduction).

+obligation(Ag,_,done(_,slowDownProduction,Ag),_)
     : .my_name(Ag) & availablePlates(N) & N <= 1
    <- stopProduction;
       goalAchieved(slowDownProduction).
```

The first plan is triggered when a delay is notified by the supplier. The robot can, then, request an account about the remaining stockpile to the feed belt. The request is performed by setting the requesting goal `requestRemainingStock` as achieved. The organizational infrastructure will issue an obligation to the feed belt to pursue the accounting goal `notifyRemainingStock`. The following Listing shows the excerpt of the feed belt robot's code, that is triggered by such obligation. The information is provided to the rotary robot by the organizational infrastructure as a new belief in its internal state.

```
+obligation(Ag,_,done(_,notifyRemainingStock,Ag),_)
     : .my_name(Ag) &
       inventory(I) & .member(plates(N),I)
    <- giveAccount(availablePlates(N));
       goalAchieved(notifyRemainingStock).
```

When the information is available, the robot will execute the **treatment goal** `slowDownProduction`.

Motor Break

Another kind of perturbation amounts to a failure in the achievement of a goal. Let us consider the elevating rotary table: it is moved by two motors, one elevating and one rotating it. Should a malfunction occur, causing the failure of goal `turnTableMoveUp`, the rotary robot would ask the table which motor stopped (`requestStoppedMotorNumber` paired by `notifyStoppedMotor` on the table's side) to notify the personnel (`scheduleTableMotorFix`) and, thus, quicken the restoration; it will, then, pause the production cell. The following accountability template captures this behavior.

```
Account Template id: at2
  concerns: turnTableMoveUp
  requesting goal: requestStoppedMotorNumber
        condition: failure(turnTableMoveUp)
  accounting goal: notifyStoppedMotorNumber
         argument: motorNumber
  treatment goal: scheduleTableMotorFix
      condition: true
```

Risk for Human Being

When a human operator enters in the operational area of the press, this, which can sense the

human presence, changes pace, slowing down for the sake of safety. The rotary robot may need to know the reason for this slowdown, eventually deciding a complete stop in case the the human gets too close to the press.

```
Account Template id: at3
  concerns: movePressDownUp
  requesting goal: requestSlowdownReason
        condition: warning
  accounting goal: explainSlowdownReason
         argument: slowdownCode
         argument: humanCoords-x
         argument: humanCoords-y
  treatment goal: emergencyStop
       condition: humanCoords-x <= 2 AND
                  humanCoords-y <= 3
```

In presence of a human operator, the rotary robot will also ask his/her position, to calibrate the functioning of the whole production cell, possibly arriving at a complete stop:

```
1  +warning(movePressDownUp)
2      <- !investigatePressSlowdown.
3
4  !investigatePressSlowdown
5      <- goalAchieved(requestSlowdownReason).
6
7  +obligation(Ag,_,done(_,emergencyStop,Ag),_)
8       : .my_name(Ag) & humanCoords(X,Y)
9      <- pressEmergencyStop;
10        activateAlarm;
11        delimitArea(X,Y);
12        goalAchieved(emergencyStop).
```

Remarks

From a programming point of view, the model has been integrated within JaCaMo's organization management infrastructure, and the runtime behavior of its normative engine (which issues obligations) has not been substantially changed. When perturbations (co-)occur, the system produces obligations accordingly, thus activating requesting, accounting, and treatment goals, exploiting JaCaMo's native ability to tackle the presence of many goals. The plans for pursueing such goals will be carried out concurrently. By way of accountability, an agent can receive accounts about perturbations occurring sparsely in the system. The agent can then decide the best treatment(s) to apply, relying on these accounts. The treatment can even involve the participation of several agents to cope with the cumulating effects of these perturbations. This is a consequence of seeing accountability as a means for gathering information and integrating efforts. When executions compete it is necessary to foresee appropriate policies at agent-program level.

Accountability is supported at no significant additional computational cost w.r.t. "standard" JaCaMo organizations; the scalability of the approach is, then, strictly related to the scalability of the underlying platform. In [15], the authors discuss the issue of building scalable JaCaMo applications through CArtAgO, which is the component that implements the organizational artifacts. Indeed, in CArtAgO, artifacts, as well as agents, can be (physically) distributed in multiple workspaces over a network. In addition, the authors propose a distribution mechanism mapping every element of a JaCaMo MAS to a web resource, following the REST architecture. The proposed model allows to describe (and access) large-scale systems as a dynamic and structured web of resources in a seamless manner.

Discussion and Related Works

We claimed that accountability is instrumental for realizing distributed systems in which robustness emerges as a design property. We have presented a conceptual model of organizational accountability. Such a model describes what accountability is, and how it relates to organizational concepts such as norms, responsibilities, tasks. By providing accounts to the agents responsible for their treatments, we capture a wide range of scenarios in which system robustness is achieved by adapting to changing or stressful conditions. Many works underline the relevance of accountability as a design concept. In Sociotechnical Systems, accountability plays a fundamental role in balancing the principals' autonomy [12]. Accountability does not limit autonomy, since a principal can decide to violate any expectation for which it is accountable; however, the principal would be held to account about that violation. A system can take advantage of a norm violation, and see it as an opportunity for innovation [16]. If the explanations, that violators are expected to give, hint a lack in the organization, the organization is updated. Although we share this vision, our proposal differs from [16], which does not foresee an explicit distinction between responsibility and accountability. The two concepts are merged within a single notion somehow aligned with liability. In our approach, the notion of accountability is not tied to liability, but has a wider understanding, since an accountable agent

is not necessarily one to be blamed. For this reason, we separate the responsibility of action from the accountability about situations [17]. We do so by introducing the structural dimension, by which the accountability of one may rely on the accountability of another. Thus, an account giver is not necessarily liable. Grounding the structural dimension on the assumption of responsibility allows agents to report legitimately about outcomes brought about by other agents [7]. This is essential when, in a distributed system, the perturbation detected by an agent may have to traverse many agents before reaching the one capable of handling it. Note that the structural dimension assures an agent has control over a task, inspired to what is introduced in [18]. The structural dimension grounds on responsibilities that, as claimed in [19], allow establishing the requirements of a system.

ReMMo (Responsibility MetaModel) [17] conceptualizes how responsibility can be structured in the frame of an enterprise architecture. Both ReMMo and our proposal agree that accountability refers to the obligation to report the achievement, maintenance, or avoidance of some given state to an authority. ReMMo relates a responsibility to an aggregate of accountabilities: a responsibility is composed of duties, and an agent assigned to that responsibility is answerable, via accountabilities, for these duties. In our case, the relationship between the two concepts is more articulated, including the structural dimension of accountability. Moreover, in ReMMO accountability overlaps with liability. In our view, it is restrictive to see accountability just as a way to find a culprit to be sanctioned; rather, it is an important tool to get a better understanding of what is going on in the system, and act accordingly. Although they may serve as deterrent, sanctions remove accountability [16]: by paying its sanction, an agent needs no longer to provide an account about its violation.

MOCA [7] is another attempt to capture accountability in computational terms. It is not a conceptual model, but an information model that captures what kind of data (facts) must be available to permit the identification of account-givers, in any situation of interest arising in a group of interacting agents.

Acknowledgements

S. Tedeschi was supported by "Bando Talenti della Società Civile" promoted by Fondazione CRT with Fondazione Giovanni Goria.

REFERENCES

1. M. J. Wooldridge, *Introduction to multiagent systems*. Wiley, 2002.
2. D. L. Alderson and J. C. Doyle, "Contrasting views of complexity and their implications for network-centric infrastructures," *IEEE Trans. Syst. Man Cybern. Part A*, vol. 40, no. 4, 2010.
3. M. Baldoni, C. Baroglio, and R. Micalizio, "Fragility and robustness in multiagent systems," in *EMAS*, ser. LNCS, vol. 12589. Springer, 2020, pp. 61–77.
4. A. W. Rawls, "Harold Garfinkel, Ethnomethodology and Workplace Studies," *Organization Studies*, vol. 29, no. 5, pp. 701–732, 2008.
5. M. J. Dubnick and J. B. Justice, "Accounting for accountability," 2004, Meeting of the American Political Science Association. [Online]. Available: https://pdfs.semanticscholar.org/b204/36ed2c186568612f99cb8383711c554e7c70.pdf
6. M. Bovens, "Two concepts of accountability: Accountability as a virtue and as a mechanism," *West European Politics*, vol. 33, no. 5, pp. 946–967, 2010.
7. M. Baldoni, C. Baroglio, K. M. May, R. Micalizio, and S. Tedeschi, "MOCA: An ORM MOdel for Computational Accountability," *J. of Intelligenza Artificiale*, vol. 13, no. 1, pp. 5–20, 2019.
8. O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with JaCaMo," *Science of Computer Programming*, vol. 78, no. 6, pp. 747 – 761, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016764231100181X
9. M. J. Dubnick, "Blameworthiness, Trustworthiness, and the Second-Personal Standpoint: Foundations for an Ethical Theory of Accountability," (September 11, 2013). Available at SSRN: http://dx.doi.org/10.2139/ssrn.2324724
10. R. W. Grant and R. O. Keohane, "Accountability and Abuses of Power in World Politics," *The American Political Science Review*, vol. 99, no. 1, 2005.
11. V. Dignum, F. Dignum, and J.-J. Meyer, "An agent-mediated approach to the support of knowledge sharing in organizations," *The Knowledge Engineering Review*, vol. 19, no. 2, pp. 147–174, 2004.
12. A. K. Chopra and M. P. Singh, "From social machines to social protocols: Software engineering foundations for

sociotechnical systems," in *Proc. of the 25th Int. Conf. on WWW*, 2016.

13. B. Burgemeester and J. Hulstijn, "Designing for the values of accountability and transparency," in *Handbook of ethics, values, and technological design : Sources, theory, values and application domains*. Springer, 2015, pp. 303–333.
14. D. H. Rached, "The Concept(s) of Accountability: Form in Search of Substance," *Leiden Journal of International Law*, no. 29, p. 317–342, 2016.
15. A. Ricci, A. Ciortea, S. Mayer, O. Boissier, R. H. Bordini, and J. F. Hubner, "Engineering Scalable Distributed Environments and Organizations for MAS," in *Proc. of the AAMAS 2019*, IFAAMAS, p. 790–798
16. A. K. Chopra and M. P. Singh, "Sociotechnical Systems and Ethics in the Large," in *Proc. of the 2018 AAAI/ACM AIES 2018*. ACM, 2018, pp. 48–53.
17. C. Feltus, "Aligning access rights to governance needs with the responsability metamodel (remmo) in the frame of enterprise architecture," Ph.D. dissertation, University of Namur, Belgium, 2014.
18. E. Marengo, M. Baldoni, C. Baroglio, A. Chopra, V. Patti, and M. Singh, "Commitments with regulations: reasoning about safety and control in REGULA," in *Proc. of AAMAS 2011*, vol. 2, pp. 467–474.
19. A. K. Chopra, F. Dalpiaz, F. B. Aydemir, P. Giorgini, J. Mylopoulos, and M. P. Singh, "Protos: Foundations for engineering innovative sociotechnical systems," in *Proc. of IEEE RE 2014*, pp. 53–62. [Online]. Available: https://doi.org/10.1109/RE.2014.6912247

Matteo Baldoni (matteo.baldoni@unito.it) is an associate professor at the University of Torino. He got a Ph.D in Computer Science and his research interests are artificial intelligence, multiagent systems, and social computing.

Cristina Baroglio (cristina.baroglio@unito.it) is an associate professor at the University of Torino. She has a Ph.D in Cognitive Sciences and is interested in engineering multiagent systems and accountability.

Roberto Micalizio (roberto.micalizio@unito.it) is an assistant professor at the University of Torino. He has a Ph.D in Computer Science and is interested in planning, model-based diagnosis, and multi-agent systems.

Stefano Tedeschi (stefano.tedeschi@unito.it) is a Ph.D. candidate at the University of Torino. His Ph.D thesis is about multiagent systems, organizations, and exceptions.