# Engineering Multiagent Organizations by Accountability and Responsibility

Matteo Baldoni[1] [0000-0002-9294-0408] (✉), Cristina Baroglio[1], [0000-0002-2070-0616], Olivier Boissier[2], [0000-0002-2956-0533], Roberto Micalizio[1], [0000-0001-9336-0651], and Stefano Tedeschi[1] [0000-0002-9861-390X]

[1] Università degli Studi di Torino - Dipartimento di Informatica, Torino, Italy
firstname.lastname@unito.it
[2] Laboratoire Hubert Curien UMR CNRS 5516, Institut Henri Fayol, MINES Saint-Etienne, Saint-Etienne, France
Olivier.Boissier@emse.fr

**Abstract.** Basing upon multi-agent systems, we show that an explicit representation of accountabilities and responsibility assumptions gives the right abstractions for properly specifying who should provide feedback to whom, about the execution of its duties, both at the level of design and at the level of programming. For the latter, we explain a programming pattern for developing accountable agents, and illustrate the approach in JaCaMo. The paper takes into account business processes as a motivating scenario.

**Keywords:** Accountability · Responsibility · BPMN · JaCaMo.



## 1 Introduction

Multiagent Systems (MAS), and in particular MAS organizations (MAO), are an approach to the design and development of complex, distributed software. In this respect, they are promising candidates to supply the right abstractions for developing business processes. In fact, a business process (BP) is "a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal." [32]. In general, a business goal is achieved by breaking it up into sub-goals, which are distributed to a number of actors. Each actor carries out part of the process and depends on the collaboration of others to perform its task.

In order to provide the right support to BPs, however, MAOs are still lacking a systematic way to properly handle feedback of the execution, provided in terms of good or bad functioning. Feedback will generally be of interest to (and should be handled by) an agent which is not the one that produces it. Therefore, an appropriate "infrastructure" needs to be devised. In [2] a proposal was made to introduce accountabilities and responsibility relationships inside a MAO. Building upon that work, here we explain how such concepts can be used as tools to systematize and guide the design and development of the agents. We also present a programming pattern, providing an exemplification in JaCaMo MAOs [8], as a

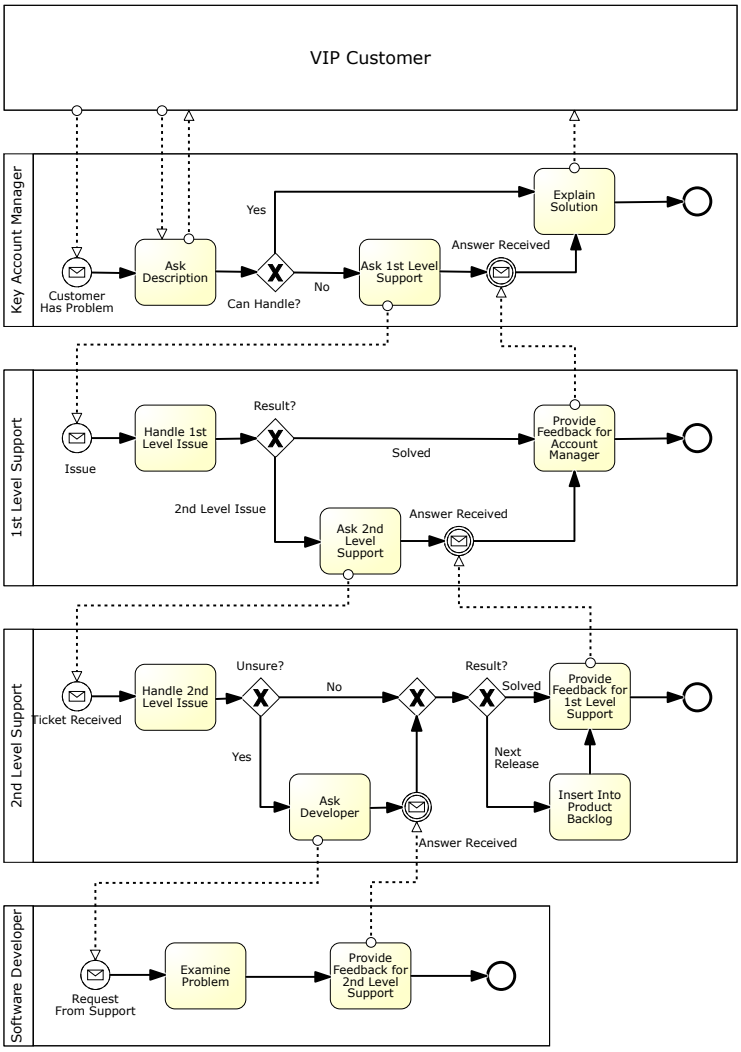


**Fig. 1.** The incident management BPMN diagram.

practical way to implement the feedback loop. This paper discusses some results that have been published in [3] at AAMAS 2019, as extended abstract[3].

The paper is organized as follows. Section 2 broadly introduces the concepts of accountability and responsibility as tools for engineering agents in MAOs to effectively realize BPs. To illustrate, we rely on the OMG Incident Management process, as a practical use case. Section 3 presents the actual programming pat-

[3] The same results have been presented in more detail, as a full paper, at the 7th International Workshop on Engineering Multi-Agent Systems (EMAS 2019), co-located with AAMAS 2019. The paper also won the Best Paper Award.

tern to develop accountable agents. In Section 4 we show how the pattern can be applied, in the scope of the JaCaMo framework, to the Incident Management scenario. Conclusions end the paper.

## 2 Responsibility and Accountability as Engineering Concepts

Figure 1 shows OMG's Incident Management BP [25] that we use as running example to exemplify interaction issues rising when feedback produced by an actor should be reported to the most appropriate one. The process models the interaction between a customer and a company for the management of a problem reported by the customer to a Key Account Manager, who, on the basis of her experience, can either solve the problem directly, or ask for the intervention of first-level support. The problem is, then, recursively treated at different support levels until, in the worst case, it is reported to the software developer. Thus, the business aim of the process (to solve the reported problem) will generally be decomposed and may be distributed over up to five BPMN processes, whose execution requires interaction and coordination. Noticeably, as usual with BPs, the way in which goals are achieved matters: agents are expected not only to fulfill their assigned goals, but also to respect the BP – the "goal" is that the process takes place [1].

One common limitation of the kind of modularity implemented both by BPs and by MAOs is that the overall process structure of the goal is intended mainly as a way for constraining the agents' autonomy, and not as information provided to support the agents in their work: agents are expected merely to focus on the achievement of the assigned sub-goals. In this way, however, the agents loose sight of the overall process and *ignore the place* their achievement has within the organization. The relationship between each level of support and the following one, in the example, is emblematic: when a request of support is made, an answer containing some kind of feedback on the realization of this support is expected in order to proceed. However, since processes are independent, one cannot give for granted that another will answer. It follows that when a process does not answer, the waiting one may get stuck indefinitely. Similarly, MAOs (e.g., [11, 14]) allow a designer to structure the functional decomposition of complex, organizational goals, and the assignment of subgoals to agents. The coordinated execution of subgoals is often supported by a normative specification, by which the organization issues obligations towards the agents, e.g., [13, 17, 12, 8]. However, agents may operate in ways that do not fit into the process specification and, importantly, when agents fail, the organization has no a systematic mechanism for sorting out the abnormal conditions, i.e., for a redress.

This is where accountability [18, 19, 10, 4, 5, 7, 6] and responsibility [29, 31, 33, 6] come in handy. Accountability "emerges as a primary characteristic of governance where there is a sense of agreement and certainty about the legitimacy of expectations between the community members." [15]. In particular, accountability implies that some actors have the right to hold other actors to

a set of standards, to judge whether they have fulfilled their responsibilities in light of these standards, and to impose sanctions if they determine that these responsibilities have not been met [19]. It presupposes a relationship between power-wielders and those holding them accountable, where there is a general recognition of the legitimacy of (1) the operative standards for accountability and (2) the authority of the parties to the relationship (one to exercise particular powers and the other to hold them to account).

Concerning responsibility, [31] proposes an ontology relating six different responsibility concepts (capacity, causal, role, outcome, virtue, and liability), that capture: doing the right thing, having duties, an outcome being ascribable to someone, a condition that produced something, the capacity to understand and decide what to do, something being legally attributable. In the context of Information Systems, the meta-model ReMMO [16] represents responsibility as a unique charge assigned to an agent. This perspective emerges also in the literature cited above, where most of the authors acknowledge that responsibility aims at conferring one or more obligation(s) to an actor (the responsibility owner). In this light, accountability is grounded on perceived/assumed responsibility, deriving from recognition of the claim-right to hold the responsible for a task to account for that very same task.

BPs represent an agreed behavior, introduce expectations on the behavior of the interacting parties, and require some kind of governance in order for the process to be enacted, but the accountability results hidden into some kind of collective responsibility ("many hands problem" [23]). As a consequence, the system is fragile to unexpected situations. When, instead, accountability relationships are expressed explicitly, they enable the reporting of unwanted conditions (i.e., account, feedback) to the most apt agent to handle them.

## 3 Engineering MAOs with Accountability/Responsibility

We refer to a setting where the organizational goal is functionally decomposed and the resulting sub-goals are assigned to agents. Since MAOs often rely on norms in order to orchestrate the executions of their agents, we imagine obligations, that are issued by the organization, to be the means through which the coordinated execution of the agents is regulated. Agents, of course, are autonomous and their fulfillment of the obligations cannot be given for granted.

Following [2], we denote by $\mathsf{R}(x, q)$ and $\mathsf{A}(x, y, r, u)$ responsibility and accountability relationships. $\mathsf{R}(x, q)$ expresses an expectation on any agent playing role $x$ on pursuing condition $q$ ($x$ is entitled and should have the capabilities of bringing about $q$); $\mathsf{A}(x, y, r, u)$ expresses that $x$, the account-giver (a-giver), is accountable towards $y$, the account-taker (a-taker), for the condition $u$ when the condition $r$ (*context*) holds[4]. We see $u$ in the context of $r$ as the agreed standard which brings about expectations inside the organization. The proposal in [2]

[4] We rely upon *precedence logic* [28], an event-based linear temporal logic, which deals with occurrences of events along runs. Event occurrences are assumed to be non-repeating and persistent. The logic has three primary operators: '$\vee$' (choice), '$\wedge$'

suggests to complement the functional decomposition of the organizational goal with a set of accountability and responsibility specifications. A set of accountability relationships is denoted as $\mathbf{A}$, and is called an *accountability specification*. The organization designer will generally specify a set of alternative legitimate accountability specifications which is denoted by $\mathbb{A}$. Given a set of accountability specifications $\mathbb{A}$, and a set of responsibility assumptions $\mathbf{R}$ (responsibility distribution), the organization is properly specified when the *accountability fitting* "$\mathbf{R}$ *fits* $\mathbb{A}$" (denoted by $\mathbf{R} \rightsquigarrow \mathbb{A}$) holds [2].

We now discuss one possible use of these concepts at *design time* for realizing accountable agents, that is, agents that provide an account of their conduct both when they succeed in achieving their goals, and when, for some reason, they fail in the attempt. Since accountabilities and responsibilities imply some obligations [19], we can realize them in JaCaMo by relying on the deontic primitives that such framework provides: the fitting relationship represented by each pair $\langle R(x, q), A(x, y, r, u)\rangle$ in $\mathbf{R}_x \rightsquigarrow \mathbf{A}_x$ (fitting projection), can be mapped into a number of Jason plans of the agent playing role $x$ by way of the following pattern, expressed in AgentSpeak(ER) [27]:

```
+!be_accountable(x, y, q) <: drop_fitting(x, y, q) {
      // Well-Doing e-plan
      +obligation(x, q) : r ∧ c <- body_q.
      // Wrong-Doing e-plan
      +oblUnfulfilled(x, q) : r ∧ c' <- body_f.
}
```

It is requested that the following conditions hold: 1) $body_q$ satisfies the *fitting-adherence* condition (below); 2) $body_f$ includes sending an explanation for the failure from $x$ to $y$. The two e-plans encode the proactive behavior of an agent assuming a responsibility. Until the responsibility is not dropped, the agent starts reacting to obligations in accordance to the accountability specified in the fitting. The agent will perceive, through the identity that is provided by the organizational role it plays, certain events as events it should tackle through some behavior of its own, but it will also be aware of its social position both (1) by knowing some other agent will have the right, under certain conditions, to ask for an account and (2) by including specific behavior for building such an account.

*Well-doing e-plan* – It is triggered when the specified obligation is issued by the MAO. The context $r \wedge c$ is satisfied when condition $r$ (activating the agent accountability) holds together with some (possibly empty) local condition $c$ that allows choosing among alternative plans, i.e., multiple ways to achieve a same result in different (local) circumstances. $body_q$ must be such to satisfy a *fitting-adherence* condition, that is, given the responsibility assumption represented by the pair $\langle R(x, q), A(x, y, r, u)\rangle$, there must exist an execution of $body_q$ that, restricted to the events that are relevant for the progression of $u$, is an actualization

(concurrence), and '$\cdot$' (before). The *before* operator allows constraining the order in which two events must occur.

of the responsibility $q$ (see [2]). Intuitively, $q$ is actually used for fulfilling the obligation. In this case, in certain applications the obligation to give an account for the satisfaction of the obligation may be implicitly resolved by satisfying the very same obligation. It is interesting to note that the accountability fitting is not only a functional specification of the organization, but it also specifies the "good" behavior of the agents. It is in fact this characteristic that justifies our programming pattern, that enriches the standard JaCaMo.

*Wrong-doing e-plan* – It allows the agent to provide an account when it did not complete its task. The triggering event, *oblUnfulfilled*, is generated by the MAO when an obligation is left unsatisfied. The context has the same structure as above; $body_f$ produces an account of the failure. This will be an explanation that the agent produces and that some other agent will use to manage the exception and to resume the execution. The correct use of the pattern guarantees, by design, that exceptional events, when occurring, are reported to the agents who can handle them properly. Accountability fulfills this purpose because, by nature, it brings about an obligation on the a-giver to give an account of what it does.

## 4 JaCaMo Accountable Agents

The engineering of accountable JaCaMo MAOs involves several steps: (1) Each process is mapped to an organizational role; (2) A scheme representing the overall process goal is defined – its successful execution amounts to the achievement of such a goal; (3) For each activity to be performed in sequence, a subgoal is added to the scheme by means of the corresponding operator; (4) For each structured block including a concurrent execution, the corresponding goals, grouped by the parallel operator, are added to the scheme; (5) For each choice all the schemes representing the possible courses of action should be defined. They will be instantiated dynamically by the agents, depending on their internal choices.

By applying the steps to Incident Management we obtain the five roles: customer ($c$), key account manager ($am$), first level support ($fls$), second level support ($sls$), and developer ($dev$). For each incident to manage, we assume there will be exactly one agent playing each role. The organizational goal is distributed into a set of schemes. The top-level scheme involves $c$ and $am$, and is instantiated by the customer $c$ when some need arises. In other terms, the scheme instantiation corresponds to the occurrence of a $report\text{-}problem_c$ event. When a problem is reported, the account manager is expected to perform $ask\text{-}description_{am}$ (ask for a description of the problem), and $c$ is expected to send what requested ($send\text{-}description_c$). Then, $am$ should provide a solution: to this aim, it can take two alternatives courses of action, both leading to the same join point of the BPMN diagram. The first alternative amounts to the case in which $am$ can handle the problem directly, and does not require the execution of any action before the join point. The second alternative, instead, amounts to the case in which $am$ cannot handle the problem directly and brings to the instantiantion of a new scheme. The agent will make a request ($ask\text{-}support\text{-}fls_{am}$) that $fls$ will manage

$\boldsymbol{a_1}$ : A($am$, $c$, $\textit{report-problem}_c$, $\textit{report-problem}_c \cdot \textit{ask-description}_{am}$)
$\boldsymbol{a_2}$ : A($am$, $c$, $\textit{report-problem}_c \cdot \textit{ask-description}_{am} \cdot \textit{send-description}_c$,
$\textit{report-problem}_c \cdot \textit{ask-description}_{am} \cdot \textit{send-description}_c \cdot \textit{explain-solution}_{am}$)
...

$\boldsymbol{r_1}$ : R($am$, $\textit{ask-description}_{am}$) $\boldsymbol{r_2}$ : R($am$, $\textit{explain-solution}_{am}$) ...

**Fig. 2.** Some accountabilities and responsibility assumptions for *Incident Management*.

either directly or by involving the next level of support – by instantiating a further scheme.

Figure 2 reports an excerpt of an accountability specification $\mathbf{A}_{incident}$ for incident management. Accountabilities $a_1$ and $a_2$ concern $am$ as a-giver. Accountability $a_1$ states that $am$ is accountable towards $c$ for asking for a description of the incident, after a problem is reported. This requirement means that $c$ can legitimately expect that, by reporting a problem to $am$, it will be asked for a description of the problem. The second accountability has a similar structure, and states that once the description of the problem has been provided, $am$ should, in the end, explain the solution, no matter what happens in the meanwhile.

With the accountability specification $\mathbf{A}_{incident}$ as a basis, the designer can identify a suitable responsibility distribution which fits it. An excerpt of an acceptable one, concerning $am$, is also reported in Figure 2.

We, now, briefly explain the realization of the key account manager $am$ agent obtained by exploiting the pattern presented in the previous section. For each pair in $\mathbf{R}_{am} \rightsquigarrow \mathbf{A}_{am}$, a g-plan must be defined, containing the proper *well-doing* and *wrong-doing* e-plans. Let us consider, in particular, the fitting involving $r_2 \rightsquigarrow a_2$. It is implemented by the following plans:

```
1  +!be_accountable(Ag,ATaker,What)
2       : .my_name(Ag) & play(ATaker,customer,incident_group) &
3         (satisfied(...,explain_solution) = What |
4          done(...,explain_solution,Ag) = What)
5      <: drop_fitting(Ag,ATaker,What) {
6
7  +obligation(Ag,_,What,_)
8       : .my_name(Ag) & can_handle(What) &
9         goalState(sch1,ask_description,_,_,satisfied) &
10        goalState(sch1,send_description,_,_,satisfied) &
11     <- goalAchieved(explain_solution).
12
13 +obligation(Ag,_,What,_)[artifact_id(ArtId)]
14      : .my_name(Ag) & not can_handle(What) &
15        goalState(sch1,ask_description,_,_,satisfied) &
16        goalState(sch1,send_description,_,_,satisfied)
17     <- createScheme(sch2, scheme2, _);
18        ?goalState(sch2,provide_feedback_am,_,_,satisfied);
19        goalAchieved(explain_solution).
20
21 +oblUnfulfilled(O)
22      : .my_name(Ag) &  obligation(Ag,_,What,_) = O & can_handle(What) &
23        goalState(sch1,ask_description,_,_,satisfied) &
24        goalState(sch1,send_description,_,_,satisfied)
25     <- .send(ATaker, tell, operation_failed_error).
26
27
```

```
28 +oblUnfulfilled(O)
29      : .my_name(Ag) & obligation(Ag,_,What,_) = O & not can_handle(What) &
30        goalState(sch1,ask_description,_,_,satisfied) &
31        goalState(sch1,send_description,_,_,satisfied) &
32        not goalState(sch2,provide_feedback_am,_,_,satisfied) &
33     <- .send(ATaker, tell, please_call_again).
34
35 +cancel-fls-request
36      : oblUnfulfilled(O) & obligation(_,_,What,_) = O
37     <- .send(ATaker, tell, please_call_Again);
38        .drop_all_intentions.
39
40 ...
41
42 }
```

The outer g-plan is triggered when the agent proactively decides to adhere to the fitting $r_2 \rightsquigarrow a_2$. Once triggered, the g-plan will remain active until the agent does not drop the fitting (see Line 5). The plans in braces encode the reactive behavior corresponding to the *well-doing* and *wrong-doing* e-plans. The first two plans, in particular, realize the *well-doing* part of the pattern. The plans are triggered as soon as an obligation to explain the solution to the customer's problem is issued (Line 7). The obligation's object (*What*) is the satisfaction of the organizational goal *explain_solution*. Indeed, in JaCaMo, the achievement of an organizational goal fulfills the corresponding obligation. Then, as requested by the pattern, the contexts of both plans must include the conditions specified in $a_2$. In JaCaMo we represent these conditions in terms of schemes that were instantiated and in terms of organizational goals that were achieved. To respect the fitting-adherence condition, both plans for *well-doing*, thus, need to include some actions that amount to $\textit{explain-solution}_{am}$, corresponding to the achievement of the given organizational goal. This is trivially true in the example (see Lines 11 and 19). The agent modifies the organizational environment thereby constructing a sequence of facts that forms the account for the specific goal.

The presence of two plans triggered by the same obligation reflects the internal choice inside the business process, driven by some local condition. The first plan is executed when *am* can handle the problem directly. The second plan, instead, is executed when the agent decides to ask for support. In this case, before providing a feedback to the customer the agent will create an instance of the second social scheme (Line 17). The successful scheme completion will provide it with a feedback from *fls*: *am* can legitimately expect such a feedback by virtue of another accountability (not reported here), in which it is a-taker. The feedback, in turn, will enable the agent to execute $\textit{explain-solution}_{am}$. The third and fourth plans, instead, deal with the *wrong-doing* part of the pattern. Should, for any reason, the obligation be unfulfilled, the agent, by virtue of its accountabilities, must provide a motivation about the violation of the obligation that binds it to the account-taker. The plan at Line 21, in particular, is triggered when the obligation is unfulfilled because of a reason that is internal to the *am* agent itself. In the fourth plan the obligation becomes unfulfilled because *am* is still waiting for a feedback from *sls*. In both cases, a proper message encoding the explanation for the failure is sent to the a-taker. The last plan, in turn exemplifies how *am* behaves as an a-taker when receives the account of a failure from

another (a-giver) agent. Specifically, the plan handles a possible failure coming from *fls* raised when it has not satisfied its obligation to provide a feedback. Event `cancel-fls-request` corresponds to the message *fls* sends as an account of such a failure, and *am* handles such a failure by asking the customer to call another time and dropping its current intention(s).

Notably, considering the accountability specification as a requirement, the actual implementation of the system results more robust. to be accountable, *am* must be capable, on the one side, of capturing exceptions from other agents (specifically, *fls*), and on the other side, of providing an account to its a-taker (i.e., the customer).

## 5 Conclusions

The systematic application of the proposed pattern makes agents aware of the process as characterization of the goal. Hence, accountabilities provide the programmer with a behavioral specification the agent has to satisfy. Our approach is specular to [33], whose objective is to determine whether a group of agents can be attributed the responsibility for a goal. Once the responsibility can be attributed to the agents, their accountability is implicitly modeled in the plan that has been inferred. Here, instead, we aim at developing agents that, by construction, satisfy the organization specification.

An interesting evolution of the present work goes in the direction of an agent-oriented type checking. Having an explicit organization model, in terms of accountabilities and responsibilities, it would be possible to devise a type checking system that verifies whether, at role enacting time, an agent possesses all the necessary plans for role playing. Moreover, since agent-based approaches are typically declarative, future work will include the exploration of other alternatives to BPMN process diagrams to depict the collaborative workflows. Declarative approaches are also used in the information systems area, in particular for what concerns business process representation. Indeed, OMG has recently released the issue 1.1 of the document for the specification of Case Management Model and Notation (CMMN) [24], which is an extension and refinement of the GSM declarative model [21]. Further investigations of these approaches will be the objective of future research. In particular, CMMN exploits an event-condition-action language that bears similarities to the way in which agents are programmed when using Jason. Still another orthogonal approach is the RALph graphical notation [9], which focuses more on the assignment of human resources to BP activities rather than on the strict definition of a workflow.

As an impact, the proposal moves MAOs closer to other paradigms where exceptions are handled, like the actor model [20], for instance, where actors that cannot handle an anomalous situation can report exceptions to their parent actors for management, and so forth further up. In an agent-based system such a scheme is not directly applicable because agents are independent entities, and show no parent-child relationship. Approaches for modeling exceptions in a MAS setting have been proposed (see, e.g., [22, 30, 26]). However, no consen-

sus has been reached on the use of such a concept in agent systems. The main problems rise when trying to accomodate the usual exception handling semantics with the properties of MAS; namely autonomy, openness, heterogeneity, and encapsulation. Accountabilities provide adequate support to fill this gap.

## References


1. Adamo, G., Borgo, S., Di Francescomarino, C., Ghidini, C., Guarino, N.: On the notion of goal in business process models. In: Ghidini, C., Magnini, B., Passerini, A., Traverso, P. (eds.) AI*IA 2018 - Advances in Artificial Intelligence - XVIIth International Conference of the Italian Association for Artificial Intelligence, Trento, Italy, November 20-23, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11298, pp. 139–151. Springer (2018). https://doi.org/10.1007/978-3-030-03840-3_-11, https://doi.org/10.1007/978-3-030-03840-3_11
2. Baldoni, M., Baroglio, C., Boissier, O., May, K.M., Micalizio, R., Tedeschi, S.: Accountability and Responsibility in Agents Organizations. In: Miller, T., Oren, N., Sakurai, Y., Noda, I., Savarimuthu, T., Son, T.C. (eds.) PRIMA 2018: Principles and Practice of Multi-Agent Systems, 21st International Conference. pp. 403–419. No. 11224 in Lecture Notes in Computer Science, Springer, Tokyo, Japan (October 31st–November 2nd 2018), http://dx.doi.org/10.1007/978-3-030-03098-8_16
3. Baldoni, M., Baroglio, C., Boissier, O., Micalizio, R., Tedeschi, S.: Engineering business processes through accountability and agents. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019. pp. 1796–1798. International Foundation for Autonomous Agents and Multiagent Systems (2019)
4. Baldoni, M., Baroglio, C., May, K.M., Micalizio, R., Tedeschi, S.: Computational Accountability. In: Chesani, F., Mello, P., Milano, M. (eds.) Deep Understanding and Reasoning: A challenge for Next-generation Intelligent Agents, URANIA 2016. vol. 1802, pp. 56–62. CEUR, Workshop Proceedings, Genoa, Italy (December 2016), http://ceur-ws.org/Vol-1802/paper8.pdf
5. Baldoni, M., Baroglio, C., May, K.M., Micalizio, R., Tedeschi, S.: An Information Model for Computing Accountabilities. In: Ghidini, C., Magnini, B., Passerini, A., Traverso, P. (eds.) AI*IA 2018: Advances in Artificial Intelligence, XVII International Conference of the Italian Association for Artificial Intelligence. Lecture Notes in Computer Science, vol. 11298, pp. 30–44. Springer, Trento, Italy (November 20th–23th 2018), https://doi.org/10.1007/978-3-030-03840-3_3/
6. Baldoni, M., Baroglio, C., May, K.M., Micalizio, R., Tedeschi, S.: Computational Accountability in MAS Organizations with ADOPT. Applied Sciences **8**(4) (2018)
7. Baldoni, M., Baroglio, C., Micalizio, R.: Goal Distribution in Business Process Models. In: Ghidini, C., Magnini, B., Passerini, A., Traverso, P. (eds.) AI*IA 2018: Advances in Artificial Intelligence, XVII International Conference of the Italian Association for Artificial Intelligence. Lecture Notes in Computer Science, vol. 11298, pp. 252–265. Springer, Trento, Italy (November 20th–23th 2018), https://doi.org/10.1007/978-3-030-03840-3_19
8. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming **78**(6), 747 – 761 (2013). https://doi.org/http://dx.doi.org/10.1016/j.scico.2011.10.004, http://www.sciencedirect.com/science/article/pii/S016764231100181X

9. Cabanillas, C., Knuplesch, D., Resinas, M., Reichert, M., Mendling, J., Ruiz-Cortés, A.: RALph: A Graphical Notation for Resource Assignments in Business Processes. In: Advanced Information Systems Engineering. pp. 53–68. Springer International Publishing (2015)
10. Chopra, A.K., Singh, M.P.: From social machines to social protocols: Software engineering foundations for sociotechnical systems. In: Proc. of the 25th Int. Conf. on WWW (2016)
11. Corkill, D.D., Lesser, V.R.: The Use of Meta-Level Control for Coordination in Distributed Problem Solving Network. In: Bundy, A. (ed.) Proceedings of the 8th International Joint Conference on Ar tificial Intelligence (IJCAI'83). pp. 748–756. William Kaufmann, Los Altos, CA (1983)
12. Dastani, M., Tinnemeier, N.A., Meyer, J.J.C.: A programming language for normative multi-agent systems. In: Handbook of Research on Multi-Agent Systems: semantics and dynamics of organizational models, pp. 397–417. IGI Global (2009)
13. Dignum, V.: A model for organizational interaction: based on agents, founded in logic. Ph.D. thesis, Utrecht University (2004), published by SIKS
14. Dignum, V.: Handbook of research on multi-agent systems: Semantics and dynamics of organizational models (2009)
15. Dubnick, M.J., Justice, J.B.: Accounting for accountability (September 2004), https://pdfs.semanticscholar.org/b204/36ed2c186568612f99cb8383711c554e7c70.pdf, annual Meeting of the American Political Science Association
16. Feltus, C.: Aligning Access Rights to Governance Needs with the Responsability MetaModel (ReMMo) in the Frame of Enterprise Architecture. Ph.D. thesis, University of Namur, Belgium (2014)
17. Fornara, N., Viganò, F., Verdicchio, M., Colombetti, M.: Artificial institutions: a model of institutional reality for open multiagent systems. Artificial Intelligence and Law **16**(1), 89–105 (2008). https://doi.org/10.1007/s10506-007-9055-z
18. Garfinkel, H.: Studies in ethnomethodology. Prentice-Hall Inc., Englewood Cliffs, New Jersey (1967)
19. Grant, R.W., Keohane, R.O.: Accountability and Abuses of Power in World Politics. The American Political Science Review **99**(1) (2005)
20. Haller, P., Sommers, F.: Actors in Scala - concurrent programming for the multicore era. Artima (2011)
21. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Fenno F. Terry Heath III, Hobson, S., Linehan, M.H., Maradugu, S., Nigam, A., Sukaviriya, P.N., Vaculín, R.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: Proc. of the Fifth DEBS. pp. 51–62. ACM (2011). https://doi.org/10.1145/2002259.2002270, http://doi.acm.org/10.1145/2002259.2002270
22. Mallya, A.U., Singh, M.P.: Modeling exceptions via commitment protocols. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 122–129. AAMAS '05, ACM (2005)
23. Nissenbaum, H.: Accountability in a computerized society. Science and Engineering Ethics **2**(1), 25–42 (1996)
24. Object Management Group (OMG): Case Management Model and Notation (CMMN), Version 1.1. OMG Document Number formal/2016-12-01 (http://www.omg.org/spec/CMMN/1.1/PDF) (2006)
25. Object Management Group (OMG): BPMN Specification - Business Process Model and Notation (2018), http://www.bpmn.org/, online, accessed 08/11/2018
26. Platon, E., Sabouret, N., Honiden, S.: An architecture for exception management in multiagent systems. Int. J. Agent-Oriented Softw. Eng. **2**(3), 267–289 (2008)

27. Ricci, A., Bordini, R.H., Hübner, J.F., Collier, R.: AgentSpeak(ER): An Extension of AgentSpeak(L) improving Encapsulation and Reasoning about Goals. In: AAMAS. pp. 2054–2056. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM (2018)
28. Singh, M.P.: 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. pp. 907–914. ACM (2003)
29. Sommerville, I., Lock, R., Storer, T., Dobson, J.: Deriving information requirements from responsibility models. In: Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009. Proceedings. pp. 515–529 (2009)
30. Souchon, F., Dony, C., Urtado, C., Vauttier, S.: Improving exception handling in multi-agent systems. In: Software Engineering for Multi-Agent Systems II. pp. 167–188. Springer Berlin Heidelberg (2004)
31. Vincent, N.A.: Moral Responsibility, Library of Ethics and Applied Philosophy, vol. 27, chap. A Structured Taxonomy of Responsibility Concepts. Springer (2011)
32. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007)
33. Yazdanpanah, V., Dastani, M.: Distant group responsibility in multi-agent systems. In: PRIMA 2016: Princiles and Practice of Multi-Agent Systems - 19th International Conference, Phuket, Thailand, August 22-26, 2016, Proceedings. pp. 261–278 (2016). https://doi.org/10.1007/978-3-319-44832-9_16