Accountability and Responsibility in Business Processes via Agent Technology

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

Università degli Studi di Torino — Dipartimento di Informatica
c.so Svizzera 185, I-10149 Torino (Italy)
firstname.lastname@unito.it

Abstract. Business processes are widely used to capture how a service is realized or a product is delivered by a set of combined tasks. It is a recommended practice to implement a business goal through a single business process; in many cases, however, this is impossible or it is not efficient. The choice is, then, to split the process into a number of interacting processes. In order to realize this kind of solution, the business goal is broken up and distributed through many "actors", who will depend on one another in carrying out their tasks. We explain, in this work, some weaknesses that emerge in this picture, and also how they would be overcome by introducing an explicit representation of responsibilities and accountabilities. We rely, as a running example, on the Hiring Process as described by Silver in [30].

Keywords: Accountability · Responsibility · BPM · MAS.

1 Introduction

Business processes are the backbone upon which enterprises rely to accomplish their business mission. A business process can be defined as "a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal." [36] It is thus not surprising that, over the last decades, many attempts of formally specifying business processes were proposed in literature. Some of these formalisms, such as BPMN, BPEL, Petri Nets, are procedural in nature and hence specify the steps (i.e., activities) that bring to the satisfaction of an initial request. Other approaches, such as DECLARE [24] and *ConDec* [23], specify a process declaratively by making explicit the relationships between tasks in terms of (temporal) constraints. All these formalisms place the concept of business activity (i.e., task) at the core of the process representation.

We think, however, that such a perspective is inadequate for capturing the relationships that exist between the actors that participate into a process, especially when the process is distributed. The most obvious scenario is in cross-organizational business processes, where different enterprises cooperate for the achievement of their business goals. In such a case, it is not possible to specify the cross-organizational process entirely because the inner processes of an enterprise (and hence its activities) are usually

hidden to its partners. A common solution is, thus, to rely on choreographies. Nevertheless, the opacity of the actual processes raises several issues about the compliance verification and the localization of flaws when something unexpected occurs. Choreographies are not sufficient to explicitly capture neither the responsibilities that an enterprise takes, nor the (legitimate) expectations each enterprise has on the others. To complicate the picture, a business process could be distributed and assigned to different actors even inside a single organization.

In this paper we claim that when business processes are by their nature distributed, a modeler should be equipped with proper abstractions for capturing relationships between the actors (or business roles), and not only between the process activities. These relationships should be based upon two fundamental concepts in human organizations: *responsibility* and *accountability*. The two terms are strictly related, and often used interchangeably in the literature, but in our perspective they assume distinct meaning and purpose. Responsibility, in particular, can assume several nuances depending on the reference context [25]. As we will explain –see also the information model described in [5]–, accountability is characterized by two fundamental facets: the legitimate expectation that an actor has on the behavior of another actor; and the *control* over the condition for which one is held to account. The intuitive meaning, here, is that an actor declares (or accepts) to be accountable for some condition when it has the capability of bringing about that condition on its own, or when it holds others to account for that same condition. In both cases the actor has control over the condition – direct in the former case, indirect in the latter. Indeed, an actor, who can hold another to account, is empowered of a form of authority, which gives it the right of asking the second actor an account of its actions about a condition of interest. Accountability, thus, plays a twofold role. On the one side, it is the instrument through which responsibilities are discharged [16]. On the other side, it is the trigger that makes interactions progress as agents tend to discharge their duties lest being sanctioned.

Moreover, we claim that approaches and technologies from the research area on multiagent systems (MAS) can be very helpful to realize this vision, finding direct application in the business processes research field. In order to explain our proposal, we rely on a reference example, whose first specification can be found in [7].

In Section 2 we introduce a motivating example expressed in BPMN. Then, in Section 3 we provide a characterization of accountability and responsibility. The following sections provide an accountability-based representation of the example and a description of an implementation.

2 A Motivating Example

As a motivating and running example we will use the Hiring Process scenario introduced by Silver [30] to exemplify the challenges of the one-to-many pattern. Suppose a case consisting of one open job position; the goal is to hire a new employee for that job. Many candidates will likely apply. As long as the position remains open, each interested candidate walks through an evaluation process, that may take some time to be completed. When a candidate is deemed apt for the position, the job is assigned and the position is closed. Silver explains how such a procedure cannot be modeled as a

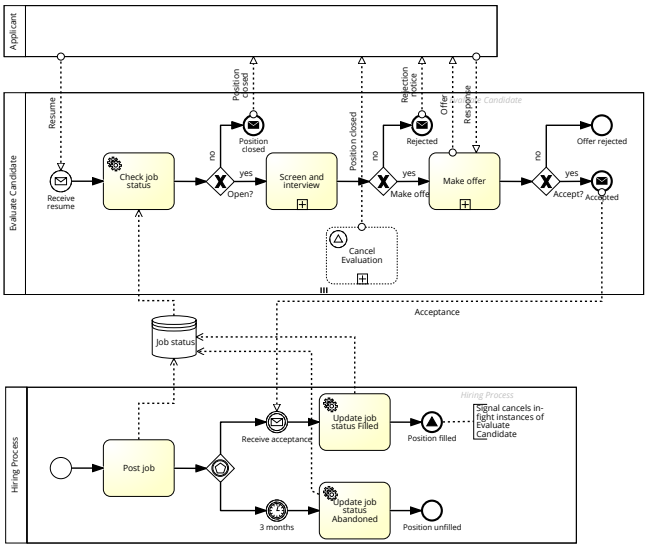

Fig. 1. The Hiring Process example: correct solution.

single BPMN process since, inherently, the activities managing the candidates (e.g., accepting and processing their applications), have a different multiplicity than the activity managing the position, which is only one.

In order to deal with many candidates for a single job, so as to meet the goal, Silver suggests the adoption of two distinct BPMN processes, that, although potentially performed by the same people, are formally represented in two separate pools since *independent processes*. These two processes are: the *Hiring Process*, that manages the job position by opening and assigning it, and the *Evaluate Candidate* process, which examines one candidate. The two processes are represented in independent pools because their respective instances do not have a 1:1 correspondence: the hiring process runs just once for a position, whereas the evaluation process runs for each candidate who shows up for the job. Possibly, many instances of this process run in parallel depending on the number of available evaluators. A coordination problem now rises because, as soon as one of the candidates fills the position, all the evaluations still in progress must be stopped. The *Evaluate Candidate* processes, thus, although processing different candidate applications, are all synchronized on the status of the position. Such a synchronization can only be guaranteed by introducing a *data storage* (Figure 1), external to the processes and accessible to all of them.

This simple yet meaningful example shows how business processes could be distributed even within a single organization. Moreover, the example offers also the opportunity to emphasize the drawbacks of an activity-centric representation, like BPMN. In fact, the relationships between the three actors are just loosely modeled via message exchange, but there is no explicit representation of the responsibilities each of them takes as a party of the interaction. For instance, the relationship between the candidate and her evaluator is emblematic: when the evaluator makes an offer to the candidate,

the evaluator expects an answer (either acceptance or rejection), but since the internal process of the candidate is hidden, we cannot give for granted that the candidate will answer. The candidate's process could be ill-defined and never answer to an offer, or, by choice, the candidate's process designer could have decided that an answer is provided only in the positive case, and not always as the evaluator expects to. It follows that when the candidate does not answer, the evaluator's process gets stuck indefinitely. What is actually missing, thus, is an explicit declaration from each party of the interaction, that they are aware of their duties and of the relationships they have with others. However, to support such declarations, duties and relationships have to be modeled explicitly. We believe that the notions of *responsibility* and *accountability* serve this purpose in an intuitive, yet effective way, and in the rest of the paper we discuss how to achieve this result.

3 Responsibility and Accountability in MAS

In this section, we characterize the concepts of responsibility and of accountability as they were introduced for multiagent systems [16, 5, 2], that is, in a way that is functional to its use in supporting the functioning of an agent system.

Concerning *responsibility*, Feltus [16] sees it as a charge, concerning a unique business task, assigned to an agent, which is always linked at least to one accountability. This view is compatible with the triangle model of responsibility by Schlenke *et al.* [29], according to which the term bears two main understandings, each of which investigated at depth by philosophers: one amounting to causation (who did it?), the other to answerability (who deserves positive or negative treatment because of the event?). Schlenke *et al.* explain how responsibility, as individuals perceive it, depends on the strength of three linkages, each of which involves two out of three basic elements, that are prescriptions, events, and identities. Prescriptions come from regulative knowledge and (broadly speaking) concern what should be done or avoided. Events simply occur in the environment. Identities include but are not limited to roles of the individual that are relevant in the context. The three linkages, thus, respectively capture whether and to which extent: a prescription is considered to concern an event, an event is considered relevant for an identity, a prescription is considered to concern an identity.

Accountability [5, 2], instead, is understood as representing the simple concept of one agent holding another to account for its actions, both "good" and "bad." In particular, the focus is on on agents that are expected to act in a certain way, and that will be held to account for that expectation's fulfillment (sometimes called positive accontability). From this definition, we identify two integral pieces to accountability: 1) a relationship between two entities in which one feels a liability to account for his/her actions to the other; and 2) a process of accounting in which actions are declared, evaluated, and scored. Accountability has distinctive traits which do not allow making it a special kind of responsibility. First of all it involves two agents, the one who gives the account and the one who takes the account [29, 21, 26, 9]. The account taker must have some kind of authority on the account giver [28, 12]. The origin of such an authority may be various; for instance, it may be due to a principal-agent relationship, or to a delegation. The account taker is sometimes called the *forum* [9]. Accountability may also

involve a sanction, as a social consequence of the account giver's achievement or non-achievement of what expected, and of its providing or not providing an account [16]. We now report from [5] the key aspects of accountability, that we rely upon together with the reference literature:

a) *Accountability implies agency.* If a principal does not possess the qualities to act "autonomously, interactively and adaptively," i.e. with agency, there is no reason to speak of accountability, for the agent would be but a tool, and a tool cannot be held accountable [31].
b) *Accountability requires but is not limited to causal significance.* The plain physical causation (also called scientific causation in [29]), that does not involve awareness or choice, does not even create a responsibility, let aside accountability. This view is supported also by [9, 10].
c) *Accountability does not hinder autonomy.* Indeed, accountability makes sense because of autonomy in deliberation [1, 29, 34, 10].
d) *Accountability requires observability.* In order to make correct judgments, a forum must be able to observe the necessary relevant information. However, in order to maintain modularity, a forum should not observe beyond its scope. For example, if a principal buys a product and the product is faulty, that principal holds the factory as a whole accountable. The factory, in turn, holds one of its members accountable for shoddy production. In other words, accountability determination is strictly related to a precise context. In each context, the forum must be able to observe events and/or actions strictly contained in its scope and decipher accountability accordingly. As context changes, accountability will change accordingly. For this reason, a mechanism to compose different contexts and decide accountability comprehensively is essential.
e) *Accountability requires control.* Without control decisions cannot be enacted, the agent does not have an impact on the situation. It will be ineffectual. In [20], control is defined as the capability, possibly distributed among agents, of bringing about events. Due to our focus on positive accountability, i.e. on bringing about a situation of interest, we follow this proposal and interpret omissions (not acting) as non-achievements. [13] gives a slightly different definition of control as the ability of an agent to maintain the truth value of a given state of affairs. Alternatives where control amounts to interference or constraint can be devised but are related to negative accountability.
f) *Accountability requires a mutually held expectation.* Accountability is a directed social relationship concerning a behavior, that serves the purposes of sense-making and coordination in a group of interacting parties, sharing an agreement on how things should be done [17]. The role of expectation is widely recognized [17, 34, 1]. Both parties must be aware of such a relationship for it to have value (the account-taker to know when it has the right to ask for an account, the account giver to know when towards whom it is liable in case of request).
g) *Accountability is rights-driven.* One is held accountable by another who, in the context, has the claim-right to ask for the account. Particularly relevant on this aspect the understanding of accountability that is drawn in tort law [12], where the account-taker is the only recognized authority who can ask for an account, and the

account-giver has a liability towards the account-taker (to explain when requested). Further analysis is carried out in [18].

In the following we use the notations $\mathsf{A}(x, y, r, u)$ and $\mathsf{R}(x, q)$ in order to explicitly represent accountabilities and responsibility assumptions, respectively. By $\mathsf{A}(x, y, r, u)$ we express that x, the account-giver, is accountable towards y, the account-taker, for the condition u when the condition r (*context*) holds. If we think of a process being collectively executed, we can say that when the r part of the process is done, then x becomes accountable of the u part. When u is true, x is considered to have satisfied the expectation that was put on it by exercising its control, which means that it has built a proof that can be supplied to the account-taker. A proof here is intended as a set of recorded facts, that demonstrate the achievement of the specified condition. Indeed, the account-taker can ask at any time for a proof to the account-giver, provided that r is true (in this case the accountability is detached). Such a proof of the partial execution will amount to the set of facts collected that far. Along with the execution, expectation and control will evolve and will run out with the satisfaction of the accountability, and only the final proof will be left. When, instead, u is false, the expectation is violated, and x's control failed. When r is false, instead, the accountability expires. This means that those conditions, by which the account-taker has the right of asking for an account to the account-giver, will not hold anymore; thus, there is no expectation about u and the account-taker may even lose its control over u. Instead, by $\mathsf{R}(x, q)$ we capture the responsibility assumption by x of the expression q. When q is true the responsibility is fulfilled, when it is false, it is neglected. When needed, we will denote by $\mathbf{A}$ a set of accountabilities, calling it an *accountability specification*, and by $\mathbf{R}$ a *responsibility distribution*, that is a set of responsibility assumptions.

Following [5, 2], where it is possible to find the technical details, $\mathsf{A}(x, y, r, u)$ is grounded on *control* and *expectation*. While expectation is naturally conveyed with the accountability itself, the control needs to be recursively verified on the structure of u. In fact, x controls u either directly or indirectly by relying on accountabilities by other parties. In words, we say that an accountability specification is *closed* when the account-giver displays the necessary control (see definition of accoutnability closure in [2]). Notice that an agent x "having control" does not mean the agent has itself the ability of making a condition become true in any circumstance, but that x has the possibility of realizing it. Moreover, the control relation on atomic expressions cannot be checked from the accountability specification only. The check depends on the responsibility assumption by the agent who has adopted the role.

Responsibility assumptions in $\mathbf{R}$ describe which duties agents take on when playing some roles inside an organization. From an organization designer's perspective, such duties would be captured in the simplest case through norm specification, or, in a richer form, norms would be complemented with requirements the agents have to comply with for adopting roles concerned by the norms. Still, this is on the organization side. On the agent side, obligations (per se) are received by fiat; following [15], they succeed in directing individual behavior only when they agree with the sensitivity of the individuals. Consequently, the design shows a weakness, that is due to a lack of explicit (and explicitly accepted) relationships. Our proposal fills this design gap through explicitly declared/taken responsibility assumptions and accountability relationships, which give

agents the means for reasoning about the implications of role enactment, and give designers the means for specifying organizations that show good properties [2].

For a normative organization to function well, its agents should interiorize the norms in their behavior but when can this happen in open organizations? If the agent considers a norm (say, an obligation) as a prescription concerning one of its identities (i.e., one of the roles it plays in the organization having that norm) the norm would start being something more than "given by fiat". In our proposal this can be done because, even though here we do not focus on the process, **R** can be derived from the organization norms; in some case, the norm specification could even reduce to the specification of the responsibility distribution – which duties are up to which roles. That would not, however, be enough if the same agent cannot see the connection between the prescription and some events it concerns (in our setting, the prescription would apply in a context), and also between the event and the identified identity (the context as one in which the role has control over something). It is the co-presence of the three linkages (1) to create in the agent the urge to tackle that context, abiding by the prescription, by virtue of its role, should the prescription apply; (2) that helps the designer to create organizations where role specification and goal distribution combine well.

4 Responsibility and Accountability in the One-to-Many Pattern

In this section we exemplify in the hiring process scenario how the notions of accountability and responsibility can be used to specify the interaction between the three involved roles.

4.1 Precedence Logic

As a first step, it is necessary to provide a language for expressing conditions, and subprocesses, that are ascribed as responsibilities to the roles, and that are used as conditions in the specification of accountabilities. To this aim, we rely upon *precedence logic* [33], an event-based linear temporal logic devised for modeling and reasoning about Web service composition. The interpretation of such a logic deals with occurrences of events along runs (i.e., sequence of instanced events). Event occurrences are assumed to be non-repeating and persistent: once an event has occurred, it has occurred forever. The logic has three primary operators: '$\vee$' (choice), '$\wedge$' (concurrence), and '$\cdot$' (before). The *before* operator 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. Such a language, thus, allows us to model complex expressions, whose execution needs to be coordinated as they are under the responsibility of different agents. Let e be an event. Then $\overline{e}$, 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, not both. Intuitively, complementary events allow specifying situations in which an expected event e does not occur, either because of the occurrence of an opposite event, or because of the expiration of a time deadline.

We also rely on the notion of *residuation*, inspired by [20, 33]. Residuation allows tracking the progression of temporal logic expressions, hopefully arriving to their satisfaction, i.e., the completion of their execution. The *residual* of a temporal expression

q with respect to an event e, denoted as q/e, is the remainder temporal expression that would be left over when e occurs, and whose satisfaction would guarantee the satisfaction of the original temporal expression q. Residual can be calculated by means of a set of rewrite rules. The following equations are due to Singh [33, 20]. Here, r is a sequence expression, and e is an event or $\top$. 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 $\Gamma_{e \cdot f} = \{e, \overline{e}, f, \overline{f}\}$.

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

Using the terminology in [4], we say that an event e is *relevant* to a temporal expression p if that event is involved in p, i.e. $p/e \not\equiv p$. Let us denote by $\boldsymbol{e}$ a sequence $e_1, e_2, \ldots, e_n$ of events. We extend the notion of residual of a temporal expression q to a sequence of events $\boldsymbol{e}$ as follows: $q/\boldsymbol{e} = (\ldots((q/e_1)/e_2)/\ldots)/e_n$. If $q/\boldsymbol{e} \equiv \top$ and all events in $\boldsymbol{e}$ are relevant to q, we say that the sequence $\boldsymbol{e}$ is an *actualization* of the temporal expression q (denoted by $\widehat{q}$).

4.2 Hiring Process Specification

In the hiring process scenario, it is quite natural to individuate three roles: the hirer, the evaluator, and the candidate. For each available position, then, there will be just one actor hi playing the hirer role, whereas many evaluators and candidates will be admissible. To simplify the exposition, we will assume that a candidate i will be evaluated by a specific evaluator ev_i; this does not exclude, however, that the same actor be an evaluator for different candidates. Namely, ev_i and i are role instances of roles evaluator and candidate, respectively. The first step of the hiring process specification is to identify the responsibility distribution $\mathbf{R}_{hiring}$, namely, what responsibilities are ascribed to each role. In our solution we have the following responsibility assignments: $\mathsf{R}(hi, \textit{fill position})$ we denote that the hirer is in charge of fulfilling the objective *fill position*, whereas $\mathsf{R}(ev_i, \textit{evaluate candidate})$ denotes that evaluator ev_i is in charge of the evaluation of a single candidate, finally, with $\mathsf{R}(i, \textit{follow-through application})$ we specify that every candidate has the objective to complete its application process. Note that *fill position*, *evaluate candidate*, and *follow-through application* are shortcut labels standing for temporal expressions encoding the business processes. In the following, the events of these processes are under the responsibility of the role indicated as a subscript.

Each responsibility is, then, characterized by a set of accountabilities describing how a role player can fulfill that responsibility. The accountability specification $\mathbf{A}_{hiring}$ for the hiring process is shown in Figure 2. It is interesting to study first the accountabilities that each evaluator ev_i assumes having part of the responsibilities over the goal. These accountabilities derive directly from the *Evaluate Candidate* process in Figure 1, which describes the procedure an evaluator is expected to follow. Roughly speaking, the hirer expects that the evaluator be compliant with the evaluation process (e.g., always perform *Screen and Interview* before a *Make Offer*). Moreover, the evaluator should interrupt the evaluation as soon as the position is assigned. On the other hand, a candidate submitting an application expects from an evaluator to be answered, either with a

$a_1 : \mathsf{A}(ev_i, hi, \mathsf{post\text{-}job}_{hi} \cdot \mathsf{apply}_i, \mathsf{post\text{-}job}_{hi} \cdot \mathsf{apply}_i \cdot \mathsf{evaluate\text{-}candidate}_{ev_i})$
$\mathsf{evaluate\text{-}candidate}_{ev_i} \equiv \mathsf{position\text{-}filled}_{hi} \cdot \mathsf{msg\text{-}position\text{-}closed}_{ev_i} \vee$
$\mathsf{check\text{-}position}_{ev_i} \cdot \mathsf{msg\text{-}position\text{-}closed}_{ev_i} \vee$
$(\mathsf{check\text{-}position}_{ev_i} \cdot \mathsf{screen\text{-}interview}_{ev_i} \cdot$
$(\mathsf{msg\text{-}rejection\text{-}notice}_{ev_i} \vee \mathsf{make\text{-}offer}_{ev_i} \cdot$
$(\mathsf{response\text{-}yes}_i \cdot \mathsf{accepted}_{ev_i} \vee \mathsf{response\text{-}no}_i \cdot \mathsf{offer\text{-}rejected}_{ev_i})))$

$a_2 : \mathsf{A}(ev_i, i, \mathsf{post\text{-}job}_{hi} \cdot \mathsf{apply}_i, \mathsf{post\text{-}job}_{hi} \cdot \mathsf{apply}_i \cdot \mathsf{inform\text{-}outcome}_{ev_i})$
$\mathsf{inform\text{-}outcome}_{ev_i} \equiv \mathsf{msg\text{-}position\text{-}closed}_{ev_i} \vee \mathsf{msg\text{-}rejection\text{-}notice}_{ev_i} \vee \mathsf{make\text{-}offer}_{ev_i}.$

$a_3 : \mathsf{A}(hi, ev_i, \mathsf{accepted}_{ev_j}, \mathsf{accepted}_{ev_j} \cdot \mathsf{position\text{-}filled}_{hi})$, where $ev_i \neq ev_j$.

$a_4 : \mathsf{A}(i, ev_i, \mathsf{make\text{-}offer}_{ev_i}, \mathsf{make\text{-}offer}_{ev_i} \cdot (\mathsf{response\text{-}yes}_i \vee \mathsf{response\text{-}no}_i))$

$a_5 : \mathsf{A}(hi, boss, \mathsf{open\text{-}position}_{boss}, \mathsf{open\text{-}position}_{boss} \cdot \mathsf{post\text{-}job}_{hi})$

$a_6 : \mathsf{A}(hi, boss, \mathsf{post\text{-}job}_{hi} \cdot (\mathsf{accepted}_{ev_i} \vee \mathsf{timeout\_3months}_{hi}), \mathsf{hiring}_{hi})$
$\mathsf{hiring}_{hi} \equiv \mathsf{post\text{-}job}_{hi} \cdot (\mathsf{accepted}_{ev_i} \cdot \mathsf{position\text{-}filled}_{hi}$
$\vee\ \mathsf{timeout\_3months}_{hi} \cdot \mathsf{position\text{-}abandoned}_{hi})$

Fig. 2. The accountability specification $\mathbf{A}_{hiring}$ for the *Hiring Process* scenario.

notification of rejection, or with a message of position filled, or possibly with an offer. All these considerations bring us to characterize $\mathsf{R}(ev_i, \textit{evaluate candidate})$ with the accountability relationships a_1 and a_2. Each event occurring in the expressions is adorned, as subscript, with the role that brings it about. Intuitively, accountability a_1 means that evaluator ev_i is accountable towards hirer hi for evaluating a candidate i, but only in the context where hirer has posted a position ($\mathsf{post\text{-}job}_{hi}$) and candidate i has applied for the position (apply_i), to guarantee this strict ordering, the sequence $\mathsf{post\text{-}job}_{hi} \cdot \mathsf{apply}_i$ appears both as the antecedent condition and as a prefix of the consequent condition, preceding the candidate evaluation. The same pattern is used throughout the subsequent relationships. The evaluation is encoded as the sequence of events that may occur during an evaluation according to the *Evaluate Candidate* process in Figure 1.

Accountability a_2 represents the expectation candidate i has on ev_i: in the context in which a job is posted and candidate i has applied for it, ev_i is expected to inform i with the outcome of the evaluation process, this can either be, a message with content "position closed", a rejection notification, or an offer for the job.

It is important to observe that, according to the definition of accountability closure [2], the accountability relationships are closed when the *a-giver* has the control (possibly indirect) over the consequent condition. Under this respect, a_1 is not properly founded since there are events in expression $\mathsf{evaluate\text{-}candidate}_{ev_i}$ that are not generated by ev_i. First of all, event $\mathsf{position\text{-}filled}_{hi}$ occurs when the hirer has assigned the position, in this case the evaluation process carried on by ev_i has to terminate by informing the candidate that the position is no longer available ($\mathsf{msg\text{-}position\text{-}closed}_{ev_i}$). To grant ev_i control over this event, thus, $\mathsf{R}(hi, \textit{fill position})$ must be characterized by

accountability a_3, which states that hi is accountable towards every evaluator ev_i still processing a candidate that, as soon as the position gets filled due to the acceptance event coming from an evaluator ev_j, hi will notify this change ($\mathsf{position\text{-}filled}_{hi}$). In other words, notifying that the position has been assigned is part of the responsibilities of the hirer role.

More critically, also the events $\mathsf{response\text{-}yes}_i$ and $\mathsf{response\text{-}no}_i$ are not under the control of ev_i. In case of an offer made to candidate i, ev_i awaits an answer, either $\mathsf{response\text{-}yes}_i$ or $\mathsf{response\text{-}no}_i$, from i. However, the candidate could never answer, and if this happened, the error would be ascribed to the evaluator for not having completed its process, rather than to the candidate for not having answered. Noticeably, this is exactly the scenario modeled in the BPMN processes in Figure 1. In fact, the BPMN model does not specify the internal process of the candidate, thus there is no guarantee that the candidate will ever answer to the evaluator's offer. The lack of proper abstractions for explicitly modeling interactions at the goal level, rather than just at the data level via message exchanges, makes the overall system fragile to unexpected conditions. We properly handle this issue thanks to the notions of *expectation* and *control* characterizing the accountability relationships. Since every accountability must be grounded over control, the engineer makes a_1 closed by associating accountability a_4 to $\mathsf{R}(i, \textit{follow-through application})$. a_4 means that candidate i is accountable towards ev_i for answering either $\mathsf{response\text{-}yes}_i$ or $\mathsf{response\text{-}no}_i$ in case it receives an offer from ev_i. This correctly captures the expectation of ev_i to receive an answer, and enables ev_i to control (even indirectly) all the events in the consequent condition of a_1.

Since a business goal, put in a context, is usually functional to other goals, it is reasonable to characterize the responsibility of the hirer with the accountability it has towards its boss. Accountability a_5 means that hi is accountable towards $boss$ for posting a job vacancy when $boss$ open a position for that job, this triggers the whole hiring process. Accountability a_6 models the fact that hi is accountable for managing the hiring process until either the assignment of the position to a candidate or to the abandoned. In particular, $\mathsf{timeout\_3months}_{hi}$ means $\overline{\mathsf{position\text{-}filled}}_{hi}$, that is, the complementary event of $\mathsf{position\text{-}filled}_{hi}$ representing the fact that, after a three-month period, the position is no longer assignable. Thus, when a candidate is accepted ($\mathsf{accepted}_{ev_i}$), hi is expected to assign the position to that candidate $\mathsf{position\text{-}filled}_{hi}$. Otherwise, in case the three-month period expires without the occurrence of an acceptance ($\mathsf{timeout\_3months}_{hi}$), hi is expected to abandon the position ($\mathsf{position\text{-}abandoned}_{hi}$). Here $boss$ can be thought of as an abstraction of the rest of the organization to which hirer and evaluator belong. This allows us to express the relations the hiring process has with other processes within the same organization. In fact, the hirer is not necessarily the same agent who opens a position, since the opening of a position might depend on decisions taken at the top level of an organization. The hirer, instead, is the agent who has the responsibility of managing the hiring process. The result achieved by this process will reasonably become the input for a downstream process.

Remark This example has shown how a distributed process can be specified in terms of a responsibility distribution **R** and an accountability specification **A**. It is worth noticing that the former is concerned with the specification of the requirements agents have to satisfy to play specific roles. Whereas, the the latter is focused on the coordination

aspect. This separation of concerns encourages both modularity and reuse. In fact, the accountability specifications can be defined and verified w.r.t. responsibility concerning roles independently of the actual agents that will play roles in the organization itself. The separation of concerns is at two levels. First, at the level of the distributed process specification: a given process can be characterized by several accountability specifications and several responsibility distributions. At this time, the consistency of the specification can be checked by verifying whether the accountability specifications fit with a particular responsibility distribution [2]. Second, at the level of *case* (i.e., process instance), the reuse of the same agents taking responsibilities, and being accountable, in different process specifications is supported by the fact that it is possible to verify whether process accountability specifications fit the responsibilities taken by agents, and are closed under control.

5 Implementation in the JaCaMo Platform

So far we have discussed how the notions of accountability and responsibility can be used, at design time, to specify distributed business processes. This enables forms of consistency checking (see e.g., [2]), that allows an engineer to discover coordination flaws that can be solved by properly adding accountability relationships or responsibility attributions. To be an effective instrument in the hands of the engineer, however, responsibility and accountability should also be means for implementing process coordination at runtime in a way which is compliant with the model defined at design time. Interestingly, accountabilities (as we proposed them) can be mapped, under certain conditions, into commitment-based protocols, and hence any agent-based platform supporting them (see for instance JaCaMo+ [3]), is a good candidate for implementing the coordination among the business processes.

Roughly speaking, a commitment-based protocol $\mathcal{P}$ is a set of social commitments [32] that agents can manipulate via a predefined set of operations. Formally, a commitment is denoted as $\mathsf{C}(x, y, p, q)$, meaning that agent x, the debtor, is committed towards y, the creditor, to bring about the consequent condition q in case the antecedent condition p is satisfied. A commitment thus formalizes a promise, or a contract, between the two agents. It creates a *legitimate expectation* in y that q will be brought about whenever p holds. This aspect is therefore very similar to what an accountability relationship $\mathsf{A}(x, y, p, q)$ creates. Intuitively, thus, for each accountability relationship an analogous commitment can be defined. However, the mapping is not always so trivial. As noticed in [11], commitments are just a way to express accountability relations, others are, for instance, authorizations and prohibitions. Therefore, in general, not every accountability relationship can be mapped into a commitment. In this paper, however, we have characterized accountability in terms of expectation and control, meaning that the *a-giver* is expected *to do*, or *to achieve*, a given condition. This specific characterization of accountability is indeed mappable into commitments since a commitment, implicitly, suggests that the debtor will behave so as to achieve the consequent condition.

In order to properly use commitments for representing accountability relationships, we have, however, to pay special care on how they are created and defined. First of all, a commitment can only be created by its debtor. This, in general, enables flexible

executions w.r.t. obligations since agents can decide what commitments to create depending on contextual conditions. However, this flexibility may cause some problems because responsibilities and accountabilities are not necessarily taken on voluntarily by an agent, but may be part of a role definition in an organization. In our hiring scenario, for instance, an agent playing the evaluator role is expected to satisfy its accountabilities related to the objective *evaluate candidate*; this is not a matter of the evaluator's initiative. To cope with this problem, we impose that an agent willing to play a given role r in $\mathcal{P}$ accepts, explicitly, all the commitments $\mathcal{C}_r$ in $\mathcal{P}$ that mention r as debtor. In [6] we present the ADOPT protocol as a means for an agent and an organization to create an accountable agreement upon the powers and commitments that the agent will take as a player of a specific role in the organization. A second feature of commitments is that, similarly to accountability relationships, there is no causal nor temporal dependency between the antecedent condition p and the consequent one q: q can be satisfied even before condition p.

Finally, and more importantly, in a commitment the antecedent and consequent conditions are not necessarily under the control of the creditor and debtor agents, respectively. In other words, agent x can create the commitment $\mathsf{C}(x, y, p, q)$ even though it has no control over q. This feature is again justified by the sake of flexibility, but this freedom endangers the realization of sound accountabilities. As argued above, in fact, our accountability relationships demand that the *a-giver* has control, possibly indirect, over the consequent condition. This means that also the corresponding commitments must retain the same property. Noticeably, it can be proved that when a set commitments implements a set of *closed* accountability relationships, all the commitments are *safe* as defined in [20]: "a commitment is safe for its debtor if either the debtor controls the negation of the antecedent or whenever the antecedent holds, the debtor controls the residuation of the consequent." In other words, the safety of the commitments is obtained as a side effect of the closeness of the accountability specification.

5.1 Implementing the Hiring Process in JaCaMo+

Having a specification of the hiring scenario in terms of accountability and responsibility, it is possible to come up with an implementation in JaCaMo+. In particular, business roles are mapped into Jason agents. Jason [8] is an agent programming language, where agents are expressed as ECA-like rules (Event-Condition-Actions) called *plans*.

In particular, each agent has a *belief base*, a set of ground (first-order) atomic formulas which represent the state of the world according to the agent's vision, and a *plan library*. Moreover, it is possible to specify *achievement* (operator '!') and *test* (operator '?') goals. A Jason plan is specified as:

$$triggering\_event : \langle context \rangle \leftarrow \langle body \rangle$$

where the *triggering_event* denotes the event the plan handles (which can be either the addition or the deletion of some belief or goal), the *context* specifies the circumstances when the plan could be used, and the *body* is the course of action that should be taken. JaCaMo+ extends Jason by allowing the specification of plans involving commitments.

These agents/processes operate by executing operations on *artifacts* implemented using CArtAgO [27]. Artifacts are programmable resources, that the agents can manipulate through a set of predefined operations, and that expose some *observable properties* perceived by the agents themselves. For the hiring process scenario, three kinds of artifacts are defined: the socialState artifact maintains the commitments corresponding to the accountabilities identified in the previous section. It is accessed by all the agents, and instantiated just once. Artifact posBA maintains the state of the position, it is instanced once for each available position, and is not accessed by the candidates. Finally, an artifact appBA, which is instantiated once for each candidate, keeps track of the states of the applications made by every specific candidate. For this reason, this artifact's instances are accessed only by the candidates and by the evaluators. To simplify the implementation of the example, we assume that a given candidate i is evaluated by a dedicated evaluator ev_i and that the couple shares an instance of appBA on which the two agents operate. The operations performed by the agents on the posBA and appBA artifacts have effect on the socialState, too, allowing the commitments to progress.

The pseudocode of the three processes is sketched in Figure 3 [1]. Listing 1.1, in particular, shows an excerpt of the hirer process. The first rule, at line 1, is activated when the commitment which encodes a_5 is *detached*, meaning that the boss has opened a position. In order to satisfy the commitment, the hirer then performs operation postJob on artifact posBA. After that, it starts waiting. The second rule, at line 6, again originates following the *discharge rule* pattern: the hirer has to properly react as soon as the commitment encoding a_3 gets detached, and hence hi update the position status to filled. It's important to point out that the execution of this rule would satisfy the commitment originating from a_6 as well. Moreover, it is not required that the $\mathsf{accepted}_{ev}$ operation is performed by a specific evaluator ev_i; the commitment is detached as soon as the first offer made by an evaluator is accepted. Finally, the rule at line 10 fires when, after the timeout of three months, the position is still open. In this case the commitment originating from a_6 is detached and the hirer has to mark the position as abandoned.

The evaluator code is sketched in Listing 1.2. It is a bit more sophisticated since it encompasses the whole evaluation process. Also in this case, however, the *discharge rule* pattern drives the implementation of such a process. In the first rule, the evaluator reacts to the detachment of commitment associated with a_1 when an application is sent, but the position has already been filled by another candidate. In this case a message of position closed is sent. Otherwise, if the position is not yet filled, with the rule at line 5, the evaluator checks the state of the position. If the result of the check is negative, again a message of position closed is sent. If the result is positive, the evaluator by interviews the candidate (see line 15). This corresponds to operation screenInterview performed upon artifact appBA that is shared between the candidate and the evaluator. After this operation, the evaluator comes up with a Choice, either to accept the candidate, thereby making an offer, or to reject it. This choice will, thus, activate a proper behavior, and hence the corresponding operation on appBA, see the rules at lines 21 - 25. Rules at lines 27 - 33 are, instead, used to react to a candidate's answer, either "yes" or "no", to a possible offer. Accordingly, the evaluator performs an operation on appBA so as to

[1] The full code of the example, implemented in JaCaMo+ [3], is available at http://di.unito.it/hiringaccountability.

```
+cc(hi,boss,open-position_boss,open-position_boss · post-job_hi,DETACHED)
        :  positionStatus(POSITION_OPEN)
        <- postJob[artifact_id(posBA)];
        !wait3months.

+cc(hi,ev,accepte_ev,accepted_ev · position-filled_hi,DETACHED)
        :  positionStatus(POSITION_OPEN)
        <- updateFilled[artifact_id(posBA)].

+cc(hi,boss,post-job_hi·(accepted_ev ∨ timeout_3months_hi),hiring_hi,DETACHED)
        :  positionStatus(POSITION_OPEN) &
        timeout_3months
        <- updateAbandoned[artifact_id(posBA)].
```

Listing 1.1. Hirer hi.

```
+cc(ev_i,hi,post-job_hi · apply_i,post-job_hi · apply_i · evaluate-candidate_ev_i,DETACHED)
        : positionStatus(POSITION_FILLED)
        <- positionClosed[artifact_id(AppId)].

+cc(ev_i,hi,post-job_hi · apply_i,post-job_hi · apply_i · evaluate-candidate_ev_i,DETACHED)
        :  not positionStatus(POSITION_FILLED)
        <- checkPosition[artifact_id(posBA)];
        // ... Result
        !closeOrScreen(Result).

+!closeOrScreen(Result)
        :  Result == no
        <- positionClosed[artifact_id(appBA)].

+!closeOrScreen(Result)
        :  Result == yes
        <- screenInterview[artifact_id(appBA)];
        // ... Choice
        !offerOrReject(Choice).

+!offerOrReject(Choice) : Choice == yes
        <- makeOffer[artifact_id(appBA)].

+!offerOrReject(Choice) : Choice == no
        <- rejectionNotice[artifact_id(appBA)].

+responseYes(candidate_i)[artifact_id(socialState)]
        :  cc(ev_i, hi, post-job_hi · apply_i, post-job_hi · apply_i · evaluate-candidate_ev_i, DETACHED)
        <- offerAccepted[artifact_id(appBA)].

+responseNo(candidate_i)[artifact_id(socialState)]
        :  cc(ev_i, hi, post-job_hi · apply_i, post-job_hi · apply_i · evaluate-candidate_ev_i, DETACHED)
        <- offerRejected[artifact_id(appBA)].
```

Listing 1.2. Evaluator ev_i.

```
+postJob(hi) <- apply[artifact_id(appBA)].

+cc(candidate_i, ev_i, make-offer_ev_i, make-offer_ev_i · (response-yes_i ∨ response-no_i), DETACHED)
        <- // ... Choice
        !response(Choice).

+!response(Choice) : Choice == yes
        <- responseYes[artifact_id(appBA)].

+!response(Choice) : Choice == no
        <- responseNo[artifact_id(appBA)].
```

Listing 1.3. Candidate i.

```
+cc(hi,boss,open-position_boss,open-position_boss · post-job_hi,CONDITIONAL)
        :  positionStatus(POSITION_NULL)
        <- openPosition[artifact_id(posBA)]
```

Listing 1.4. Boss $boss$.

Fig. 3. The business processes as declarative ECA rules.

progress in the evaluation process and, then, satisfy its commitment. Interestingly, it's worth noting that no specific rule is requested for treating the commitment associated with a_2 because whenever such a commitment gets detached, the evaluator satisfies it by satisfying the first one. However, at a normative layer, the commitment which encodes a_2 is fundamental to detect the misbehavior of the evaluator towards the candidate.

Listing 1.3 sketches the pseudocode of a candidate. This process applies for a position when a job is posted (see operation apply performed upon appBA), and, then, reacts to the detachment of commitment associated with a_4 by answering either "yes" or "no" to an offer.

Finally, Listing 1.4 shows the pseudocode of the boss. This process is not present in the original version of the example, but here it is necessary to start the whole, by detaching the commitment which encodes a_5. Indeed, the only action performed by the boss is to open the position as soon as the commitment associated with a_5 is created, thereby detaching it.

6 Discussion and Conclusions

Goals are the final purposes that justify every activity in business processes. Surprisingly enough, however, goals are not modeled explicitly in the standard modeling languages for business applications (e.g., BPMN). Especially in cross-organizational settings, the lack of an explicit representation of the business goal raises many problems, including documentation, design checking, and compliance of the implementation. Cross organizational business processes are often modeled via *choreographies* which define "the sequence and conditions under which multiple cooperating, independent agents exchange messages in order to perform a task to achieve a goal" [35]. Choreographies are therefore a means for reaching goals in a distributed way, but the goal is in the mind of the modeler, whereas the model itself boils down to an exchange of messages that may be not sufficiently informative. As discussed in [14], the *interconnection model* style of BPMN hampers the modularity and reuse of the processes, and creates situations where the interaction is easily ill-modeled. Decker et al. propose a novel modeling style for choreographies: the *interaction model*, and introduce iBPMN as an extension to BPMN. In iBPMN a choreography becomes a first-class component as it lays outside the business processes. Specific language elements allow the modeler to express how the interaction progresses through a workflow, where elementary activities include the (atomic) send-receive of messages, and decision and split gates. iBPMN allows the modeler to define ordering constraints between the interaction activities, and it is certainly a more powerful tool for expressing choreographies than standard BPMN, however, the goal of the interaction is still in the mind of the modeler and it is not made explicit.

In this paper we have shown how the notions of responsibility and accountability provide the engineer with explicit modeling tools, through which it becomes possible to distribute a business goal among different processes, yet maintaining precise dependencies among them via accountability relationships. For the sake of exposition, we have presented our approach through an example of the one-to-many coordination pattern. However, our proposal is applicable for cross-organization integration in the broad

sense. An explicit representation of accountability relationships has several advantages. First of all, it makes the assessment of the correctness of an interaction model possible. As shown in the simple hiring scenario, for instance, we have singled out a flaw in the BPMN model proposed by Silver, thanks to the formal characterization of control associated with an accountability relationships. Moreover, our proposal paves the way to *compatibility* and *conformance* checks. Intuitively, compatibility centers around whether a set of process models can interact successfully. Conformance, on the other hand, focuses on whether a process model is a valid refinement or implementation of a given specification [14]. We have also pointed out that our accountability relationships are not just an abstract modeling tool, but find a proper implementation in commitment-based protocols. The obvious advantage, thus, is to translate the interaction model into a compliant implementation.

Agent-based approaches are typically declarative. Declarative approaches are also used in the information systems area, in particular for what concerns business process representation. The GSM model [19] is an attempt to represent in a declarative way the artifact lifecycle. Such a goal is considered so important that recently the OMG has released the issue 1.1 of the document for the specification of Case Management Model and Notation (CMMN) [22], which is an extension and refinement of GSM. Further investigations of these approaches will be the objective of future work. In particular, CMMN exploits an event-condition-action language that bears similarities to the way in which agents are programmed when using Jason.

References

1. Anderson, P.A.: Justifications and precedents as constraints in foreign policy decision- making. American Journal of Political Science 25(4) (1981)
2. Baldoni, M., Baroglio, C., Boissier, O., May, K.M., Micalizio, r., Tedeschi, S.: Accountability and responsibility in agent organizations. In: PRIMA 2018: Principles and Practice of Multi-Agent Systems, 21st International Conference. Lecture Notes in Computer Science, Springer (2018)
3. Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Commitment-based Agent Interaction in JaCaMo+. Fundamenta Informaticae 159(1-2), 1–33 (2018)
4. Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Type Checking for Protocol Role Enactments via Commitments. Journal of Autonomous Agents and Multi-Agent Systems 32(3), 349–386 (May 2018)
5. Baldoni, M., Baroglio, C., May, K.M., Micalizio, R., Tedeschi, S.: An Information Model for Computing Accountabilities. In: Ghedini, C., Magnini, B., Passerini, A., Traverso, P. (eds.) Proc. of 17th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2018). Springer, Trento, Italy (2018), in this same volume.
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: Ghedini, C., Magnini, B., Passerini, A., Traverso, P. (eds.) Proc. of 17th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2018). Springer, Trento, Italy (2018), in this same volume.
8. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons (2007)

9. Burgemeestre, B., Hulstijn, J.: Handbook of Ethics, Values, and Technological Design, chap. Designing for Accountability and Transparency. Springer (2015)
10. Chopra, A.K., Singh, M.P.: The thing itself speaks: Accountability as a foundation for requirements in sociotechnical systems. In: IEEE 7th Int. Workshop RELAW. IEEE Computer Society (2014), http://dx.doi.org/10.1109/RELAW.2014.6893477
11. 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)
12. Darwall, S.: Morality, Authority, and Law: Essays in Second- Personal Ethics I, chap. Civil Recourse as Mutual Accountability. Oxford University Press (2013)
13. Dastani, M., Lorini, E., Meyer, J.C., Pankov, A.: Other-condemning anger = blaming accountable agents for unattainable desires. In: Proc. of AAMAS. ACM (2017)
14. Decker, G., Weske, M.: Interaction-centric modeling of process choreographies. Information Systems 36(2), 292–312 (2011)
15. Durkheim, E.: De la division du travail social. PUF (1893)
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. Garfinkel, H.: Studies in ethnomethodology. Prentice-Hall Inc., Englewood Cliffs, New Jersey (1967)
18. Grant, R.W., Keohane, R.O.: Accountability and Abuses of Power in World Politics. The American Political Science Review 99(1) (2005)
19. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., III, F.F.T.H., 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: Proceedings of the Fifth ACM International Conference on Distributed Event-Based Systems, DEBS 2011, New York, NY, USA, July 11-15, 2011. pp. 51–62 (2011)
20. Marengo, E., Baldoni, M., Baroglio, C., Chopra, A., Patti, V., Singh, M.: Commitments with regulations: reasoning about safety and control in REGULA. In: Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS). vol. 2, pp. 467–474 (2011)
21. Nissenbaum, H.: Accountability in a computerized society. Science and Engineering Ethics 2(1), 25–42 (1996)
22. 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)
23. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006, Proceedings. pp. 169–180 (2006)
24. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, Maryland, USA. pp. 287–300 (2007)
25. van de Poel, I.: The Relation Between Forward-Looking and Backward-Looking Responsibility, pp. 37–52. Springer Netherlands (2011)
26. Quinn, A., Schlenker, B.R.: Can accountability produce independence? goals as determinants of the impact of accountability on conformity. Personality and Social Psychology Bulletin 28(4), 472–483 (April 2002)
27. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective. Autonomous Agents and Multi-Agent Systems 23(2), 158–192 (2011), http://dx.doi.org/10.1007/s10458-010-9140-7

28. Romzek, B.S., Dubnick, M.J.: Accountability in the Public Sector: Lessons from the Challenger Tragedy. Public Administration Review 47(3) (1987)
29. Schlenker, B.R., Britt, T.W., Pennington, J., Rodolfo, M., Doherty, K.: The triangle model of responsibility. Psychological Review 101(4), 632–652 (October 1994)
30. Silver, B.: BPMN Method and Style, with BPMN Implementer's Guide. Cody-Cassidy Press, Aptos, CA, USA, second edn. (2012)
31. Simon, J.: The Online Manifesto: Being human in a hyperconnected era, chap. Distributed Epistemic Responsibility in a Hyperconnected Era. Springer Open (2015)
32. Singh, M.P.: An ontology for commitments in multiagent systems. Artif. Intell. Law 7(1), 97–113 (1999)
33. 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)
34. Suchman, L.: Discourse, Tools, and Reasoning: Essays on Situated Cognition, chap. Centers of Coordination: A Case and Some Themes. Springer-Verlag, Berlin (1997)
35. W3C: W3C Glossary and Dictionary (2003), /http://www.w3.org/2003/glossary/, /http://www.w3.org/2003/glossary/
36. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007)