Goal Distribution in Business Process Models

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

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 [13].

Keywords: Accountability · Responsibility · BPM · Goals.

1 Introduction

Most of the methodologies for the design and development of complex distributed systems turn around the notion of *goal* as a way to specify the functional behavior of the system under development. In the agent-oriented programming, for instance, the Gaia methodology [20] is specifically concerned with how a society of agents cooperate to realize the *system-level goals*.

The notion of goal is also relevant in the specification of business processes. Specifically, 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." [19] As a general practice, it is desirable to design an end-to-end process [13]: a single process is triggered by a specific request and ends up with a proper answer for that request (e.g., a service or a product). In other words, since the Business Process Modeling and Notation (BPMN) is one of the most widely used languages for modeling business processes, the description of how satisfying a business request should be encompassed within a single BPMN process, whenever possible, so that any instance of such a process would be a specific *case*.

Sometimes, however, such a practice cannot be implemented, for instance because the activity instances are not aligned across the whole end-to-end process. To cope with this problem, the engineer cannot but resort to model the end-to-end process as multiple BPMN processes, realizing in this way the one-to-many coordination pattern

[13, 10]. In the one-to-many pattern, the designer must model a number of independent BPMN processes that interact with each other either by exchanging messages, or by means of a synchronized access to a shared data storage. Since these processes are independent, each of them has its own start and end events, however, none of them alone can bring about the business goal. Such a goal, in fact, is only achieved when all the processes are considered as a whole. This represents a first pitfall of the one-to-many pattern because the business goal is no longer explicitly represented by a single BPMN process. Moreover, interactions among the processes may be only indirectly represented – as synchronized accesses to data storages. As a consequence, the BPMN models lose part of their descriptive power because relevant coordination aspects are actually missing. Thus, an engineer has to deal with several processes at design time, but she has not adequate abstractions to model, and check, how these processes will actually interact.

We deem that providing an engineer with explicit abstractions for capturing interactions at the level of goals is essential for the realization of complex distributed systems, and in particular, systems where multiple, concurrent business processes interact with each other to achieve a business goal. To support the achievement of goals in a distributed way, we resort to notions that underlie human coordination, namely, *accountability* and *responsibility*. The two terms are strictly related, and often used interchangeably in the literature, but in our perspective they assume distinct meaning and purpose. For instance, in [11] responsibilities are seen as charges assigned to some actor. Being responsible for a task, however, does not necessarily mean to directly carry out the very same task. A task, in fact, can be decomposed into subtasks, each of which is then under the responsibility of a different actor. Here comes into play the notion of accountability that is broadly defined as: the obligation to give account to someone else under the threat of sanction(s) [8, 14, 5]. In other works, e.g. [6], accountability is a directed relationship between two principals, which reflects the legitimate *expectations* the second principal has of the first about the achievement of a condition. Moreover, thanks to the inherent expectations created by an accountability relationship, accountability is seen as a major driving force of individuals when it comes to decide about their own behavior [1]. In our perspective, see the information model described in [3], accountability is characterized by two fundamental facets. First, as already noted, the legitimate expectation that an actor has on the behavior of another actor. Second, the *control* over the condition for which one is held accountable. The intuitive meaning, here, is that an actor declares (or accepts) to be accountable for a given condition when it has the capabilities for bringing about the condition on its own, or when it can rely on the accountability relationships provided by other actors. In both cases the actor has control over the condition –direct in the former case, indirect in the latter. In fact, an actor that can hold another to account exerts a form of authority having 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 [11]. On the other side, it is the trigger that makes interactions progress as agents tend to discharge their duties lest being sanctioned.

In this paper, we first point out some modeling issues related to the one-to-many pattern for coordinating independent business processes (Section 2). We then describe our

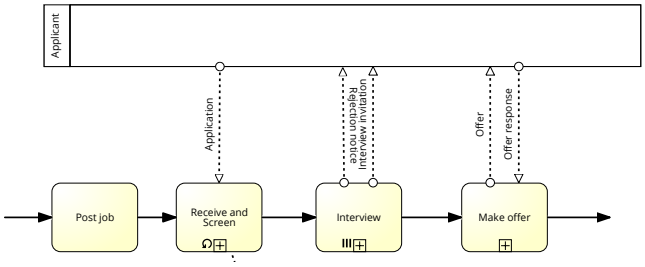

Fig. 1. The Hiring Process example: ineffective solution.

proposal (Section 3), and argue how responsibility and accountability can be accommodated in the context of business processes to keep an explicit trace of (1) how a business goal is distributed over concurrent processes, each responsible for a specific portion of the goal, and (2) how the coordination among these processes is represented via accountability relationships. Having an explicit model of goal distribution has several beneficial consequences. First of all, it becomes clear what is the (sub)goal carried out by each business process and how this is functional for the achievement of the overall goal. Moreover, having a formal model, the engineer can verify whether the processes she has designed actually achieve the overall goal, and hence detect in advance flaws. In particular, we will show (Section 4) that thanks to the recursive feature of control associated with accountability, an engineer can establish whether each process is properly supported by the others, and what is potentially missing. Finally, we show how, driven by the goal distribution model, the interaction among the processes can be implemented as commitment-based protocols, provided that some specific conditions on the creation and shape of commitments are respected (Section 5). Finally, we conclude with a comparison between our proposal and novel approaches to modeling business process choreographies (Section 6).

2 Challenges in Business Goal Distribution

To understand the issues hidden in the one-to-many pattern, we shortly recall how Silver [13] illustrates this problem by means of the Hiring Process example. 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 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. A first intuitive solution is showed in Figure 1 where the multiplicity of the candidates is dealt with by means of an iterative task (i.e., `Receive and Screen`) and a parallel task (i.e., `Interview`). Although this solution is formally

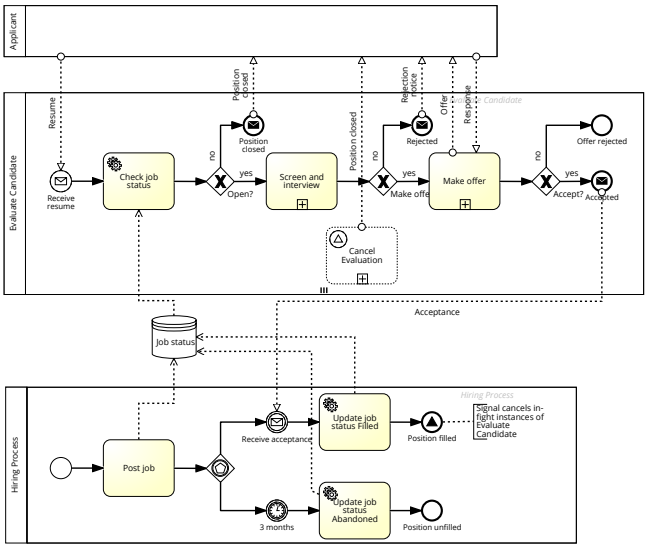

Fig. 2. The Hiring Process example: correct solution.

correct, it does not meet the goal. Indeed, the goal of the hirer is to assign the position as soon as an adequate candidate is found. The process in Figure 1, instead, awaits to collect a certain number of applications, and then starts the interviews (in parallel). A first weakness of the process is that the hirer has to set the number of applications she is willing to wait before staring the interviews. Once this number is reached, it is not possible to accept other applications. Thus, it may be the case that the position remains unfilled because none of the interviews were satisfactory and no new application is admissible. Moreover, the hirer moves to step `Make Offer` only after having interviewed all the candidates. This process, thus, does not represent the intended goal since an offer is not made as soon as a good candidate is found.

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 2), external to the processes and accessible to all of them.

We see in the solution proposed by Silver some critical issues. A first problem is that the goal *fill position* is achieved by the *Hiring Process*, but this process is cleared off any meaningful activity. The process, in fact, just opens a position in the beginning, and then awaits for an event either to assign the position or to abandon it. The hiring agent loses the control on the candidates' evaluation. When a candidate is accepted by an evaluator, the hiring agent does not know how the evaluation process has been carried out. On the other hand, in case the three-month timeout expires, the hiring agent cannot know whether all the candidates were rejected, or no candidate at all showed up. Looking at the *Evaluate Candidate* process, instead, one knows how an evaluator manages a candidate, but cannot contextualize the process: the evaluator does not know how many positions are open, nor she knows there is a time limit. The (sub)goal *evaluate candidate* is functional to the achievement of the original business goal *fill position*, but the relationship between these two goals (and processes) is hidden in the synchronized access to the data store. Data are used as a sort of synchronization signals, but then the semantics associated with data falls outside the model. Arguably, the engineer adds an annotation to the model through which she explains data semantics, but then the correct development of the processes relies on the skills of the careful developer.

Our goal, on the contrary, is to maintain the dependencies between the *Hiring* and the *Evaluate Candidate* processes in an explicit, formal way that allows one to develop them so as to be compliant with the coordination designed by the engineer. 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 Distributing Goals via Accountability and Responsibility

In order to support the achievement of distributed goals, we resort to the notions of *accountability* and *responsibility*. Specifically, we take as a starting point the ReMMo conceptual model proposed by Feltus [11], which is, to the best of our knowledge, the first proposal that relates responsibility and accountability in a computational model. ReMMo, in fact, defines responsibility as "a charge assigned to a unique actor to signify its accountabilities concerning a unique business task." Responsibility is therefore seen as a charge assigned to an agent, which is always linked at least to one accountability.

Accountability, on the other hand, has distinctive traits which do not allow making it a special kind of responsibility. It involves two agents, the one who gives the account (*a-giver*) and the one who takes the account (*a-taker*). The *a-taker* can only be someone who has some kind of authority on the account giver [7]. The origin of such an authority may be various; for instance, it may be due to a principal-agent relationship, or to a delegation. Moreover, accountability is defined into a context (or condition), and concerns either *to-do* or *to-achieve* a specific goal (or task). 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. So, for instance, both a doctor and a nurse will be responsible – in different ways – for a treatment task to be given to a patient. The different ways in which they are responsible depend on the many accountabilities they will have concerning this task (e.g., towards the head physician, towards the administration, towards the patient, the nurse towards

the doctor, ...). The nurse will be accountable *to do* (herself) certain procedures, the doctor *to achieve* (relying on others) certain results.

The ReMMo model is surely a valuable contribution, but it fails in providing a proper characterization of accountability. The notion of expectation, for instance, is just informally assumed but it is not explicitly captured within the model. Moreover, we deem that an agent can be held accountable for a given goal only when it has the *control* over the goal. Control can be defined as the the ability, possibly distributed among agents, of bringing about events [12]. In fact, without control, the agent does not have an impact on the situation. It will be ineffectual. Again, the notion of control is not part of the ReMMo model. Discussing how the ReMMo model could be enhanced with a more precise characterization of accountability is out the scope of this paper, a first attempt is discussed in [3]. In the rest of the paper, we assume that each accountability relationship brings over the two features of legitimate expectation and control.

We now illustrate in abstract terms how a business model can be complemented with accountabilities. Let us suppose that a modeler has devised a set of independent business processes that, working in a choreography, achieve a given business goal. As discussed above, the goal is clearly in the mind of the modeler, but since it has been distributed over a number of processes, in the resulting BPMN model, BP, the business goal remains substantially implicit. To make the goal explicit, first, we consider each process assigned to a role x in BP as an objective ob, whose achievement contributes to the achievement of the business goal, and, second, we say that role x is responsible for that specific objective, denoted by $\mathsf{R}(x, ob)$. In other terms, we can think of an objective as a high-level task that abstracts from execution details. The set of responsibilities make explicit what high-level tasks are needed to achieve a business goal, and which role is in charge for each task. However, the responsibilities do not model how an objective is related to another objective, that is, how it is functional to, and coordinated with, other objectives.

To this end, we follow the same idea of grounding responsibilities on accountabilities, followed also in [11], and characterize each $\mathsf{R}(x, ob)$ with a set of accountability relationships of form $\mathsf{A}(x, y, p, q)$: role y, the a-taker, is entitled to ask role x, the account-giver, for a proof about the (non-)realization of condition q, when condition p is realized. In his case, x must provide the account, that is, x will produce a sequence of events that prove the satisfaction of the expectations y has on x. Events in the proof amount to the execution of a (sub)process meeting condition q. Generally speaking, the two conditions p and q need not to be causally nor temporally related, i.e. q can be brought about even if p does not hold yet. Condition p just circumscribes the context under which x is held to account for q. When modeling the coordination of distributed business processes, however, the intuition is that q should be actually achieved after p.

More precisely, x and y are roles in BP, whereas p and q can be thought of as events representing portions of the BP process relevant for expressing the accountability relation. Intuitively, events in p and q corresponds to the sending/reception of messages, the throwing/catching of signals, and the execution of activities as indicated in BP. (At the time being, the language for expressing the antecedent and consequent conditions is not relevant in this abstract presentation, a specific formalism is introduced for a concrete scenario in the next section, where we show how the coordination between antecedent

and consequent conditions can be obtained.) However, not all the events in q may be under the direct control of x. On the other hand, we demand that the consequent condition be under the control of the a-giver. This requires an indirect form of control that is achieved only when all the accountability relationships associated with every role in BP are *closed*.

Definition 1. *Let BP be a business process achieving a given business goal, and let $\mathcal{A}$ be the set of all the accountability relationships $\mathsf{A}(x, y, p, q)$ for each pair of roles x and y in BP. We say that $\mathcal{A}$ is* closed *if for each accountability relationship $\mathsf{A}(x, y, p, q) \in \mathcal{A}$, every event e occurring in q is such that either x can bring about e directly, or there exists in $\mathcal{A}$ another accountability relationship $\mathsf{A}(z, x, p', q')$ such that e holds when q' holds (i.e., $q' \models e$), and x has direct control over p'.*

Control over condition q is therefore exerted by agent x in two ways. For *direct control* we mean that x can bring about every event in q directly via its own behaviors. For *indirect control* we mean that x can bring about some of the events in q by relying on the collaboration of other agents, that are subjected to x via accountability relationships through which x can pressure them, having the control of the antecedent condition, to pull their weight on achieving the condition q.

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

In this section we exemplify in the hiring process scenario how our accountability framework is used to model the one-to-many pattern. 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 agent be an evaluator for different candidates. Namely, ev_i and i are role instances of roles evaluator and candidate, respectively. To each role we ascribe part of the responsibility of achieving the business goal, namely with $\mathsf{R}(hi, \textit{fill position})$ 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.

Each responsibility is, then, characterized by a set of accountabilities describing how a role player can meet that responsibility. The set of accountabilities for the hiring process is showed in Figure 3, where antecedent and consequent conditions are expressed in precedence logic. [1] It is interesting to study first the accountabilities that

[1] Precedence logic is an event-based linear temporal logic introduced in [16] and in [17, Chapter 14] for Web service composition. The interpretation of such a logic deals with occurrences of events along runs (i.e., sequence of instanced events). Under this respect, event occurrences are assumed as nonrepeating and persistent: once an event has occurred, it has occurred forever. The precedence logic has three primary operators: '$\vee$' (choice), '$\wedge$' (concurrence), and '$\cdot$' (before). The *before* operator allows one to constrain 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 immediately after one another.

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

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

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

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

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

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

Fig. 3. The set of accountabilities relationships for the *Hiring Process* scenario.

each evaluator ev_i assumes having part of the responsibilities over the goal. These accountabilities derive directly from the *Evaluate Candidate* process in Figure 2, 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 notification of rejection, or with a message of position filled, or possibly with an offer. All these considerations bring us to characterize $\mathsf{R}(ev_i,$ *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 (post-job$_{hi}$) and candidate i has applied for the position (apply$_i$), to guarantee this strict ordering, the sequence post-job$_{hi} \cdot$ 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 2.

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, to be properly grounded, the accountability relationships must be such to guarantee 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 evaluate-candidate$_{ev_i}$ that are not generated by ev_i. First of all, event position-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 (msg-position-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 (position-filled$_{hi}$). In other words, notifying that the position has been assigned is part of the responsibilities of the hirer role. Note that, even if ev_i and ev_j are distinct, they are role instances of the same role evaluator, and hence a_3 satisfies the definition of indirect control in Definition 1.

More critically, also the events response-yes$_i$ and response-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 response-yes$_i$ or response-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 2. 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 properly 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 response-yes$_i$ or response-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, timeout_3months$_{hi}$ means $\overline{\text{position-filled}}_{hi}$, that is, the complementary event of position-filled$_{hi}$ representing the fact that, after a three-month period, the position is no longer assignable. Thus, when a candidate is accepted (accepted$_{ev_i}$), hi is expected to assign the position to that candidate position-filled$_{hi}$. Otherwise, in case the three-month period expires without the occurrence of an acceptance (timeout_3months$_{hi}$), hi is expected to abandon the position (position-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.

Accountabilities framework in action. It is worth noting that our proposal is not only useful at the modeling stage (e.g., to recognize flaws in the interaction among business processes), but it also provides runtime support for handling exceptions. In fact, via the accountability relationships it is possible to isolate the actors responsible for unexpected executions. Let us assume, for instance, that after a reasonable amount of time, candidate i complains that it has not received a response about its application, yet the position results assigned to a different candidate. The isolation of the actor(s) that misbehaved would in general require the examination of the processes' logs to reconstruct the actual sequence of events that has led to the missing notification. Our accountability framework, instead, does this in a more direct and effective way. If i has not received an answer, then ev_i has not satisfied a_2, that is, ev_i has not sent none of the three possible messages. Thus ev_i is surely to blame for its misbehavior. Our framework allows for a more in-depth analysis, if we detect that also a_3 has not been satisfied, then hi, too, is responsible for the unexpected behavior since event position-filled$_{hi}$ is necessary for ev_i in order to send msg-position-closed$_{ev_i}$ (see a_1). In a different scenario where hi fails in satisfying a_3 and ev_i behaves correctly, ev_i will carry out the evaluation and will issue to i either an offer or a rejection notice. This anomalous behavior is detectable and explainable by simply observing rejection notices or offers instead of "position closed" messages after that the position has been assigned.

5 Implementing Accountabilities

So far we have discussed how the notions of accountability and responsibility can be used at design time to model a one-to-many pattern of interaction among independent processes. In particular, we have shown how, taking into account the expectation and control facets of accountability, the engineer can verify, at design time, whether the interaction among the processes is well founded. To be an effective instrument in the hands of the engineer, however, responsibility and accountability should also be means for implementing process coordination in a way which is compliant with the model. Interestingly, accountabilities (as we proposed them) can be mapped, under certain conditions, into commitment-based protocols, and hence any platform supporting them (see for instance JaCaMo+ [2]), 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 [15] 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 [6], 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 use properly commitments for representing accountability relationships, we have, however, to pay special care on how these commitments 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 create depending on contextual conditions. However, this flexibility may create 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*, it 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 [4] 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 implement a set of *closed* accountability relationships, all the commitments are *safe* as defined in [12]: "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 safeness of the commitments is obtained as a side effect of the closeness of the accountability relationships.

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 set-

tings, 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" [18]. 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 [9], 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 [9]. 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.

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., Capuzzimati, F., Micalizio, R.: Commitment-based Agent Interaction in JaCaMo+. Fundamenta Informaticae 159(1-2), 1–33 (2018)
3. 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.
4. Baldoni, M., Baroglio, C., May, K.M., Micalizio, R., Tedeschi, S.: Computational Accountability in MAS Organizations with ADOPT. Applied Sciences 8(4) (2018)
5. Bovens, M.: Two concepts of accountability: Accountability as a virtue and as a mechanism. West European Politics 33(5), 946–967 (2010), https://doi.org/10.1080/01402382.2010.486119
6. 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)
7. Darwall, S.: Morality, Authority, and Law: Essays in Second- Personal Ethics I, chap. Civil Recourse as Mutual Accountability. Oxford University Press (2013)
8. Day, P., Klein, R.: Accountabilities: Five Public Services. Social science paperbacks, Tavistock (1987)
9. Decker, G., Weske, M.: Interaction-centric modeling of process choreographies. Information Systems 36(2), 292–312 (2011)
10. Dumas, M.: On the convergence of data and process engineering. In: Proc. of Advances in Databases and Information Systems. LNCS, vol. 6909, pp. 19–26. Springer (2011), https://doi.org/10.1007/978-3-642-23737-9_2
11. 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)
12. 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)
13. Silver, B.: BPMN Method and Style, with BPMN Implementer's Guide. Cody-Cassidy Press, Aptos, CA, USA, second edn. (2012)
14. Sinclair, A.: The chameleon of accountability: Forms and discourses. Accounting, Organizations and Society 20(2-3), 219–237 (1995)
15. Singh, M.P.: An ontology for commitments in multiagent systems. Artif. Intell. Law 7(1), 97–113 (1999)
16. 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)
17. Singh, M.P., Huhns, M.N.: Service-oriented computing - semantics, processes, agents. Wiley (2005)
18. W3C: W3C Glossary and Dictionary (2003), /http://www.w3.org/2003/glossary/, /http://www.w3.org/2003/glossary/
19. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007)
20. Wooldridge, M., Jennings, N.R., Kinny, D.: The GAIA methodology for agent-oriented analysis and design. Autonomous Agents and multi-agent systems 3(3), 285–312 (2000)