Journal on Data Semantics manuscript No.
(will be inserted by the editor)

Process Coordination with Business Artifacts and Multi-Agent Technologies

Matteo Baldoni · Cristina Baroglio · Federico Capuzzimati · Roberto Micalizio

Received: date / Accepted: date

Abstract This work is set in the context of data-centric approaches, and is motivated by the observation that business artifacts are not devised as natural means of coordination, despite the fact that they have this potential. Instead of using orchestration and choreography languages, we propose to enrich business artifacts with a normative layer that defines the coordination, basing our approach on social commitments. The straightforward advantage is an increased reusability of both processes and business artifacts, thanks to a clear decoupling between the coordination logic and the business logic. We show how social commitments can be leveraged for modularizing the design of distributed tasks and discuss the advantages of the approach from a software engineering perspective.

Keywords Business Artifacts · Normative MAS · Social Commitments

1 Introduction

The *artifact-centric* approach [8, 17, 14] is a viable solution for specifying and deploying business operations by combining both data and processes as first-class citizens. In particular, the notion of *Business Artifact* [25] opened the way to the development of a data-driven approach to the modeling of business operations. The data-driven approach counterposes a data-centric vision to the activity-centric vision, traditionally used when processes are explicitly modeled in terms of workflows. *Business artifacts* are business-relevant objects that are created and evolve as they pass through business operations. They include an *information model* of the data, and a *lifecycle model.* The latter captures the key states through which data evolve and their transitions, and it is used both at runtime (to track the evolution of business artifacts), and at design time (to distribute tasks among the processes that operate on a business artifact). Business artifacts, however, have a drawback: they do not provide any means for designing and modularizing the *coordination* of those processes which should operate on them, which is obtained, instead, via service choreographies [9].

These overlay on the business artifacts a sequentialization of the operations on them, but they are not capable of exploiting the information that evolves inside the business artifacts themselves. Consequently, when two processes need to coordinate their execution, in order to understand whose turn it is, it becomes necessary to add some further means of synchronization. Moreover, the processes, that use a business artifact, must encapsulate in their bodies the exchange of messages as prescribed by the choreography.

In the paper (Section 2), we explain the limits and the drawbacks of this approach, with the help of the Hiring Process example [31], and propose to enhance business artifacts in a way that allows exploiting their natural potential as a medium of coordination. We do so by introducing a *normative layer* to capture the behaviors that are expected of the parties. The Hiring Process is also used as a running example throughout the paper. We also claim (Section 3) that results from the research area on multiagent systems (MAS) support the enhancement of business artifacts as coordination media by providing conceptual (and practical) instruments. An actual implementation, that relies upon JaCaMo+ [3], is described in Section 4.

M. Baldoni · C. Baroglio · F. Capuzzimati · R. Micalizio
Università degli Studi di Torino, Dipartimento di Informatica,
Tel.: +390116706711, E-mail: firstname.lastname@unito.it

Improvement with respect to the workshop paper. This work improves the proposal [2] presented at the BPAI 2017 workshop in the following way. It introduces the one-to-many coordination problem and the Hiring Process as an instance of it, and fully develops this example from specification to implementation. It explains the issues concerning coordination in business processes (both process-centric and artifact-centric). It better explains the reasons for relying on concepts and results from the MAS literature. It develops the idea of a normative layer by introducing and characterizing coordination artifacts, by explaining the relationships between business artifacts and coordination artifacts, and by exemplifying their development and use. It consistently revises the proposed architecture. It provides a fully developed example (Hiring Process) with a link to its implementation.

2 Challenges in Process Coordination

The way for achieving a result not always can be encompassed within a single process. This happens, for instance, when the multiplicity of an activity is not in accordance with the multiplicity of another one. In this case, it is necessary to separate them into two different processes leading to the *one-to-many* pattern of coordination between the process instances [31,21]. A well-known example in literature is the *hiring scenario* illustrated by Silver [31].

Example 1 (Hiring Process [31]) A hirer opens a call for a job position for which many candidates will likely apply over a time period. As long as the position remains open, each candidate is called for an interview, and then evaluated. The evaluation of a single candidate takes time, even weeks. Thus, one would not want to process a whole application before starting to consider another. Rather, it would be better to carry on the evaluation of many candidates at the same time. For similar reasons, usually one would not want to postpone the decision on who to hire until the completion of the examination of the last candidate. Indeed, it would be better to end the selection as soon as a good candidate is identified – that is, when the business goal is achieved.

Coordination and Business Processes. Silver addresses the one-to-many problem discussing how it can be modeled by way of a set of interacting BPMN processes ([31, pages 115-118]), each pursuing a different objective. Figure 1 shows the proposed solution, consisting of a *Hiring Process*, each of whose instances tackles a single job opening, an *Evaluate Candidate process*, whose instances tackle each a different candidate, and an *Applicant Process*, whose instances amount to candidates. While the relationship between the instances of *Evaluate Candidate* and *Applicant* is *one-to-one*, the relationship between the instances of the *Hiring Process* and those of the *Evaluate Candidate* process is inherently *one-to-many*. The states of all these processes are to be coordinated and, in particular, the *Hiring Process* must have a way to enable *Evaluate Candidate* when a new job is opened, and to disable its running instances when the job position has been assigned.

The solution proposed by Silver highlights the limitations of BPMN in modeling the coordination. In fact, with reference to Figure 1, the BPMN representation only suggests that the *Hiring Process* should update the status of a job opening after receiving the acceptance of an offer by some candidate, and that this will let *Evaluate Candidate* know that it accomplished its task for that opening. Nevertheless, the language is not provided with the expressive means for capturing such a coordination explicitly. The introduction of a *data storage* which is *external* to the processes, but to which all of them have access, supports the *synchronization* of the processes [31], and the *data consistency*. However, this is not sufficient. When a business goal is split over a set of interacting processes, each of these will realize only a part of the overall goal, and, in order to realize such a part, it will generally depend on the achievement of sub-goals that are realized by other processes. The synchronization at the level of data that is realized in BPMN by the introduction of data storages does not capture coordination in such high-level terms, due to the fact that interactions among the processes can be only indirectly represented – as synchronized accesses to data storages. Thus, the BPMN models lose part of their descriptive power.

Coordination and Business Artifacts. An alternative approach to cope with the one-to-many coordination pattern is the adoption of *business artifacts*. Business artifacts add an information layer concerning both the structure and the lifecycle of the data they encompass. Some authors [21] propose to use them as a means to combine process engineering with data engineering. Still, as explained in [2], they do not support coordination satisfactorily. To understand why, let us focus on the BALSA (Business Artifacts with Lifecycle, Services, and Associations) methodology [9], that we consider as a significant representative of the business artifacts approaches. Here, coordination among of business processes is tackled by relying on *choreographies.* As an example, Figure 2 shows two business processes, bp_1 and bp_2, both defined declaratively in terms of ECA

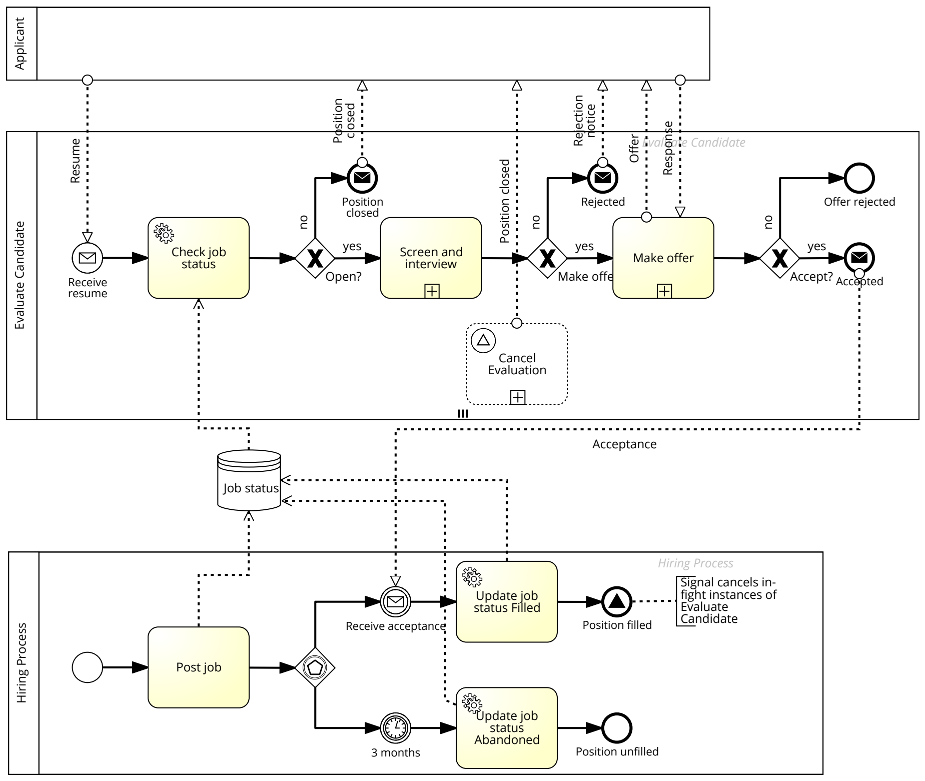

Fig. 1 The Hiring Process example represented in BPMN [31].

(Event-Condition-Action) rules, which access to a same business artifact. The fact that the bp_1 and bp_2 *comply with* the specification of a common choreography guarantees the right synchronization between the services they invoke. Choreographies realize a form of *subjective coordination* [27]. This means that each business process needs to include also the "interaction logic" of the choreography role it plays along with its business logic. Figure 3 shows a possible service choreography for the *Hiring Scenario* –for the sake of readability, the picture reports only the case of an interaction leading to filling the position. The choreography involves five components: two business artifacts, i.e., positionBA and applicationBA (which respectively model the status of a job position and the status of the application by a candidate), and three processes, i.e., the hirer hi, the candidate i, the evaluator ev_i for candidate i. Solid arrows represent operations that a process carries out over a business artifact, whereas dashed arrows represent notices about state changes sent by an artifact to the interested processes.

As explained, each involved process must comply with this choreography, so, for instance, process ev_i should perform task screen-interview only after a position is opened and an application for that position is received. Both such tasks belong to the business logic of other processes and, in principle, their occurrence does not require an involvement of ev_i. However, in order to allow the coordination among the process to occur respecting the schema encoded by the choreography, it is necessary to introduce explicit notifications that the process should receive both about the opening of a job position and about an application for that position, and such notification should occur *in the right order*. Should post-job and apply be performed in the reversed order, the evaluator should not perform screen-interview. This constraint cannot but be hard-coded in the evaluator's body –as a context condition, before starting screen-interview, the evaluator verifies whether the previous sequence of events is correct. This is, however, no little requirement because in order to implement a compliant evaluator, it is necessary not only to know which actions the evaluator should perform, and their order, but also the order in which the others will send their notifications. In other words, there is a strict coupling among the implementations of all the interacting processes. Consequently, the design and implementation become more complex, and the possibility of reusing the same process in different contexts is reduced. Furthermore, there is the need to guarantee that the middleware, and the medium through which notifications are delivered, preserve the order in which notifications

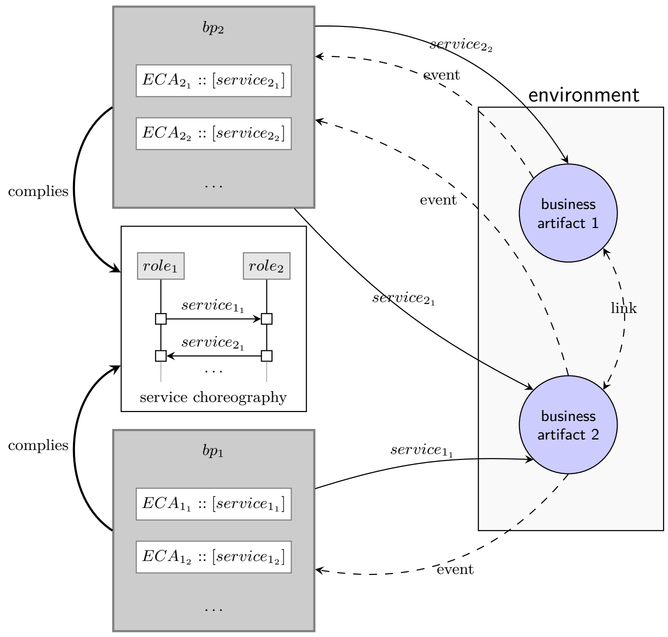

Fig. 2 Synchronized access to a business artifact via a choreography.

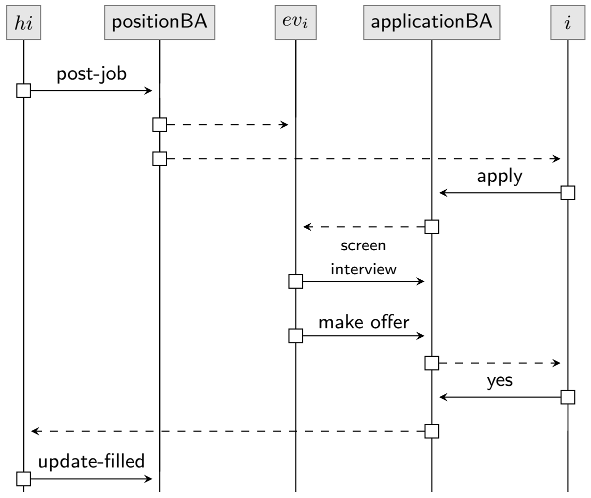

Fig. 3 A possible choreography for coordinating components in the *Hiring Scenario*.

are generated. That is, the interacting processes must be aligned [16], a condition which cannot generally be guaranteed.

Coordination in Multiagent Systems. Since the early proposals for MAS programming, organizations have been seen as metaphors for modularizing the code. Organizations, in fact, provide an overall abstraction of the task the agents have to achieve. In Gaia [40,38], for instance, organizations are characterized by two features: a set of roles and a set of interactions among roles. Here interactions are seen as protocol definitions; where a protocol is "an institutionalized pattern", namely, a pattern that has been formally defined [38]. The pattern, thus, defines the rules (i.e., *norms*) through which an institutional reality takes shape and evolves [12]. This institutional reality is the actual means of coordination of the agents: its constituent elements, the *institutional facts*, have a social meaning, by way of norms, that is shared and understood by all the agents participating to the interaction. Agents, thus, act so as to bring about those institutional facts that represent their goals or duties towards others.

In other terms, when norms are explicitly represented and known by all the participants, it is possible to create *expectations* about the behavior of others in response to given messages, and this allows determining when and how to act. Indeed, coordination is all about expectations: an activity can fruitfully be carried out by many parties when there is a clear understanding on what each one should do and when. So one party will wait for the completion of the task by another party before starting its part. On the same grounds, the party who is first to act is confident that another party will continue from where it stopped. So, for instance, assume that the hiring scenario includes the following norm: "whenever a job position is open, the evaluator is held to handle any incoming application". Knowing such a norm, the hirer opens a position because it is now a necessary step for accomplishing its goal of having a new employee. The same norm is used by the evaluator: in order not to violate the norm, it will only assess applications arrived after the position was open.

In comparison, business processes and service choreographies lack a clear management of expectations. For instance, Figure 1 suggests that the *Hiring Process* opens a position by adding an entry in a shared data store, and as a consequence it expects to close this position after receiving the acceptance of an offer by some candidate. However, this expectation is not explicitly formulated within the BPMN model, and it is only in the mind of the designer. Also service choreographies in business artifacts fall short in explicitly modeling the mutual expectations between the processes. For instance, the *Hiring Process* knows that by performing service post-job it will change the status of positionBA from vacant to open. Studying the lifecycle of the same business artifact, one comes to know that the job status can, then, evolve into filled or abandoned, but this is not sufficient to expect that it will ever progress. In fact, the progression of the job status will depend on services carried out by other processes (i.e., the candidate and the evaluator) over other business artifacts (i.e., the applicationBA). All this is only partially capture by the service choreography in Figure 3, which specifies the causal chain of communications at the level of messages, but

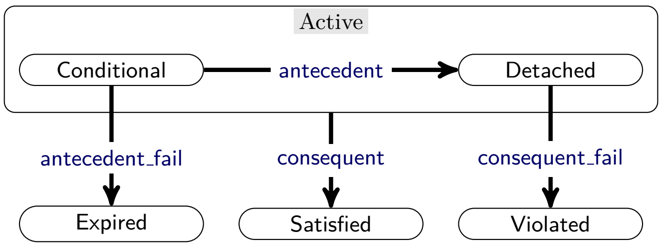

Fig. 4 Commitment life cycle [35].

provides no way for relating such communications to the evolution of information (i.e., state changes in the involved business artifacts).

3 Enabling Coordination on Top of Business Artifacts

Taking advantage of the solutions conceived in the MAS field, we present in this section our proposal for coordinating (business) processes using business artifacts complemented with a normative layer, so as to achieve a form of *objective coordination* [30]. Specifically, we express such a normative layer in terms of social commitments, thus the section starts with a short background about them. The proposal is subsequently exemplified in the Hiring Process scenario.

3.1 Background

A *social commitment* [15,32] models the directed relation between two principals: a *debtor* and a *creditor*, that are both aware of the existence of such a relation and of its current state: a commitment $\mathsf{C}(x, y, s, u)$ captures that principal x (debtor) commits to principal y (creditor) to bring about the consequent condition u when the antecedent condition s holds. Commitments are always created by their debtor and can be manipulated by means of the standard operations *cancel* (executed by debtor), *release* (by creditor), *assign* (by debtor), *delegate* (by creditor) [32].

Commitment evolution follows the lifecycle formalized in [35], which is reported in Figure 4. A commitment is *Violated* either when its antecedent is true but its consequent is false, or when it is canceled when Detached. It is *Satisfied*, when the consequent is true. It is *Expired*, when its antecedent is false (the commitment is no longer in effect). A commitment is *Active* when it is initially created. Active has two substates: *Conditional* as long as the antecedent does not occur, and *Detached* when the antecedent has occurred.

Commitments have a normative power in the sense that they bring about the expectation that the debtor will satisfy the commitment once detached.

The events that constitute the commitment conditions are always observable and all the involved principals, by observing their occurrence, can infer which commitments hold, and their state. In particular, we adopt *precedence logic* [33] to express antecedent and consequent conditions. 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. Such a language, thus, allows us to model complex expressions about the relevant events that may occur during the progression of a business process, and to consider these expressions as antecedent or consequent conditions of commitments. Intuitively, events in antecedent and consequent conditions correspond to the sending/reception of messages, the throwing/catching of signals, and the execution of activities of a business process. In the following, event_{role} denotes the business *role* that brings about event.

The works described in [20,4] propose methodologies, namely Amoeba and the 2CL Methodology, for designing the commitments involved in an interaction protocol.

3.2 Coordination Architecture

The conceptual architecture of our proposal is shown in Figure 5. The architecture shows how principals can coordinate while operating in their environment through the introduction of a *normative layer*, which amounts to the set of social commitments that can be created and evolve along with the interaction.

Principals operate in the environment by "using" business artifacts, that is, by invoking some of the services/operations made available by the business artifacts themselves. These operations act at the data level, and in some cases they can will cause a progression of the business artifact state along its lifecycle. The state of a business artifact falls into the information model layer of our picture. Thus, whenever a business artifact changes its state, this change is propagated to all the linked artifacts.

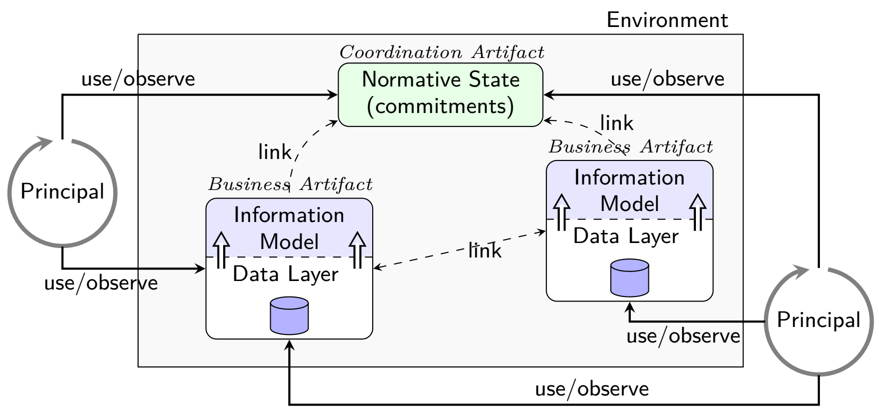

Fig. 5 Environment/Information System based on business artifacts and coordination artifacts.

In particular, we reify the normative layer by means of a specific class of artifacts, called *coordination artifacts*, which are made available to the interacting processes. Such resources allows the interacting principals to know which commitments hold; these, in turn, create expectations on the others' behavior, that is, that the debtors of commitments will behave so as to satisfy the corresponding consequent conditions. When this does not happen, a violation is detected and made available for the appropriate management. The propagation of a state change to a coordination artifact, in general, will cause the creation of new commitments or the evolution of already existing ones. This, in turn, will, on one hand, push a principal to act so as to satisfy any detached commitment in which the principal appears as the debtor, and on the other hand, push a principal who is the creditor of a conditional commitment to act so ast to detach it.

To make this discussion more concrete, let us consider a simple example, featuring a merchant m and a customer c as two interacting principles. These two principles share an environment where a business artifact payments and coordination artifact purchase are available. The first artifact traces the payments that c does for some specific goods whose price amounts to 300 euros. To this end, the artifact makes available operation pay(*amount*), through which the customer can pay for the goods. The customer can either pay the sum with a single pay operation, or pay by installments, e.g., three installments of 100 euros each. In both cases, as soon as the requested amount of 300 euros is deposited, the business artifact payments progresses into the state "paid". This is the information that is notified to the linked coordination artifact purchase. This artifact traces the state of commitment $\mathsf{C}(m, c, \mathsf{paid}_c, \mathsf{ship}_m)$, meaning that, when the goods are paid for (event paid_c occurs), the merchant m is committed to ship the goods, and hence bring about event ship_m. The point, now, is that if the customer is interested in having the goods, it will act so as to detach the commitment, by paying. To do that, the customer will act upon business artifact payments, by means of the operations made available by this artifact. As soon as the goods are paid for, this information, just local in the business artifact, assumes a social value by being mapped into an event that makes commitment progress to the detached state. The merchant is now asked to satisfy its commitment by shipping the goods.

Thus, principals act upon both business artifacts and coordination artifacts. In particular, by observing on a coordination artifact, a principal will be aware of the existing commitments and of the expectations that are yielded by them. Instead, by observing on a business artifact, principals get to know the set of operations the artifact makes available, and which transitions these operations cause, according to the artifact lifecycle. Principals can, then, act in two ways. The first way is through operations performed upon some business artifacts. Such operations impact on the data layer of the artifacts, but, as we explained, they can also cause a change in the state of the artifact information layer. The second is to act directly on the coordination artifacts by executing commitment operations, like create or cancel, with the aim of shaping the coordination [3].

In contrast to the service choreographies of the BALSA model, our architecture enables a form of *objective coordination* [27], where coordination is addressed outside the interacting principals. Objective coordination enables a clear separation between the implementations of the business logic and of the coordination logic by explicitly representing the environment where the principals operate. In our proposal, we meet this property by reifying a normative layer, expressed in terms of commitments, inside a dedicated coordination artifact, which is external to the principals. The interaction logic is thus encoded into a single component (i.e., the coordination artifact), and is not distributed and intermingled within the principals' code. A positive consequence is that the implementation of environment resources (i.e., artifacts) and of the principals' processes can be carried out and verified in *isolation*.

3.3 The Hiring Process Scenario

We exemplify the presented architecture in the scenario of the Hiring Process, that we have used to introduce the challenges of the one-to-many coordination pattern. In Figure 6, hi (hirer), ev_i (evaluator), and i (candidate) represent the principals of this scenario, that observe and use artifacts in a shared environment. For each available position, there will be just one principal playing the hirer role, whereas many evaluators and candidates will be possible. 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 principal be an evaluator for different candidates. The workspace includes also three artifacts: one coordination artifact, and two business artifacts. The coordination artifact hiringNormativeState contains the social commitments through which the normative layer is realized. It is accessed by all the principals, and instantiated just once. The business artifact positionBA, instead, maintains the state of the position, it is instantiated just once, and it is only accessed by the hirer and the evaluators. Finally, the business artifact applicationBA is instantiated once for each candidate, each instance keeping track of the status of the application that was made by that specific candidate. This artifact is, therefore, accessed only by the candidate and by the associated evaluator.

The Business Artifacts. We now explain the two business artifacts positionBA and applicationBA, whose lifecycles are shown in figures 7 and 8, respectively. positionBA is a very simple artifact, that traces the state of the position throughout the process carried out by the hirer: the nodes represent the position state, while the edge between nodes represent the possible operations that the artifact makes available to its users.

The information model of the business artifact applicationBA encompasses two elements: the state of the application and the state of the eventual offer. Each node of the graph gives an intuitive idea of how these two elements evolve as a consequence of the operations performed on them by the principals' processes. The lifecycle takes into account that the evaluation process can terminate at any stage when the position is assigned to a candidate. Indeed, it is interesting to point out that this happens when the hirer performs operation update filled on positionBA, which hence evolves to final state position filled. Notably, position filled has also an impact on the coordination artifact hiringNormativeState, making the consequent of commitment c_1 progress. Such a progression is, therefore, captured by the evaluator which should inform the candidate that the position is no longer available. This passage will be discussed in detail in the next section.

The Coordination Artifact hiringNormativeState. We now discuss in detail some critical elements of our proposal. We begin by showing how the normative layer can be expressed in terms of commitments so as to satisfy the specification of the BPMN processes in Figure 1.

Specifically, hiringNormativeState maintains the coordination model expressed by the commitments in Figure 9, where a the expectations they create are graphically represented as edges between principals. The commitment meanings are as follows.

- *Commitment* c_1. It encodes that ev_i is committed to carry out the evaluation for the application by candidate i according to a predefined procedure. The procedure, outlined in $\mathsf{evaluate\text{-}candidate}_{ev_i}$, is equivalent to the *Evaluate Candidate* process in Figure 1. The sequence $\mathsf{position\text{-}filled}_{hi} \cdot \mathsf{msg\text{-}position\text{-}closed}_{ev_i}$, describes that the evaluator informs candidate i that the position is closed as soon as the position is assigned by hi. On the BPMN process, this is equivalent to the *Check Job Status* activity and the consequent message sending in case the position has already been assigned, and it also models the capturing of signal *position filled* sent by the hirer while this evaluator is still processing an application. The rest of the condition, ($\mathsf{screen\text{-}interview}_{ev_i} \ldots \mathsf{offer\text{-}rejected}_{ev_i}$) encodes all the possible branches of the execution of process *Evaluate Candidate*, including the messages sent to and received from other roles. It is important to note the shape of the antecedent and of the consequent conditions of c_1, among which a temporal relation is captured. Indeed, ev_i's commitment is detached (and hence the principal will have to bring about the evaluation process) only when a job position has been posted, and a candidate has applied for it. In order to model that ev_i is expected to evaluate candidate i only after this has applied for the position, the antecedent condition (i.e., $\mathsf{post\text{-}job}_{hi} \cdot \mathsf{apply}_i$) occurs as a prefix in the consequent condition. A similar pattern is used also in the following commitments.
- *Commitment* c_2. Evaluator ev_i takes also commitment c_2 towards candidate i to take into account the application and to provide i with an answer for the

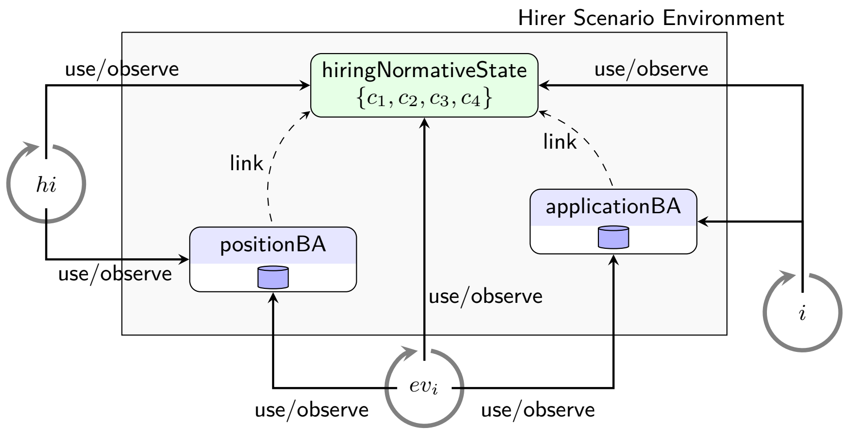

Fig. 6 The *Hiring Process* scenario implemented with business and coordination artifacts.

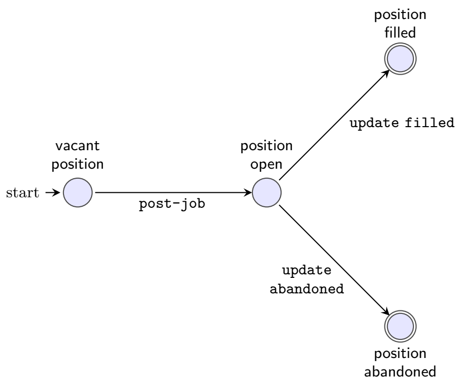

Fig. 7 The lifecycle of business artifact positionBA.

application. Such a commitment has the same antecedent condition of c_1. The answer can either be a message informing that the position has already been closed, or a rejection notice, or even an offer for the job. Also in this case, apply_i is used in order to make $\mathsf{inform\text{-}outcome}_{ev_i}$ follow the satisfaction of the antecedent condition.

– *Commitment* c_3. This commitment is pretty interesting from our point of view. It represents the candidate i's promise to answer either "yes" or "no" to an eventual offer made by ev_i. The BPMN in Figure 1 does not specify the internal behavior of the candidate, so an answer for the offer cannot be taken for granted. Indeed, the candidate could never answer, and yet the evaluator could not be able to detect this anomalous situation and would remain stuck awaiting indefinitely. In our opinion, this example highlights the weaknesses of BPMN in modeling the coordination of independent processes. Certainly, one could enrich the evaluator's process so as to wait a predefined time interval for an answer. But this is just a way for handling the exception. With commitments, instead, our major concern is to stimulate the principals to act so as to make the interaction progress. When candidate i is offered the job, it is stimulated to answer either "yes" or "no" due to the existence of commitment c_3. Any anomalous situations in which the candidate does not answer is clearly detected by the violation of c_3. The evaluator process does not need to capture this eventuality directly.

– *Commitment* c_4. Finally, commitment $c4$ represents the engagement of the hirer towards the evaluators, and in particular describes the process carried out by the hirer in Figure 1. The antecedent condition expresses the start event of the process (i.e., $\mathsf{post\text{-}job}_{ev_i}$) followed by two alternative events that enables the hirer to complete the process. The first event is accept_{ev_i}, meaning that an evaluator has found a suitable candidate. The second event is $\mathsf{timeout\_3months}_{hi}$, which stands for the complementary event to $\mathsf{position\text{-}filled}_{ev_i}$. In precedence logic it is not possible that both event e and its complementary $\bar{\mathsf{e}}$ occur along the same run of execution. Therefore, after a period of three months, when $\mathsf{timeout\_3months}_{hi}$ occurs, the position cannot be assigned anymore. The consequent condition of c_4 describes how the hirer completes the process depending on what event has satisfied the antecedent condition. In case of event accept_{ev_i}, hi assign the position and notify this to all the evaluators possibly still running ($\mathsf{position\text{-}filled}_{ev_i}$). Notably, this allows the evaluators still active to bring about event $\mathsf{msg\text{-}position\text{-}closed}_{ev_i}$ so as to discharge both commitments c_1 and c_2. In case the antecedent has been satisfied by event $\mathsf{timeout\_3months}_{hi}$, instead, the position is abandoned.

4 Implementation in the JaCaMo+ Platform

In the previous section we have provided a conceptual architecture for setting up, on top of business artifacts,

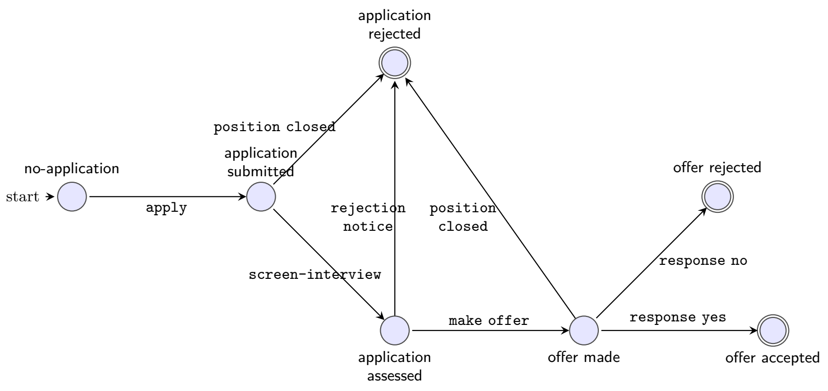

Fig. 8 The lifecycle of business artifact applicationBA.

$$\begin{tikzcd}
\boxed{hi} \arrow[r, bend left, "c_1"] & \boxed{ev_i} \arrow[l, bend left, "c_4"] \arrow[r, bend left, "c_2"] & \boxed{i} \arrow[l, bend left, "c_3"]
\end{tikzcd}$$

$$c_1 : \mathsf{C}(ev_i, hi, \textsf{post-job}_{hi} \cdot \textsf{apply}_i, \textsf{post-job}_{hi} \cdot \textsf{apply}_i \cdot \textsf{evaluate-candidate}_{ev_i})$$

$$c_2 : \mathsf{C}(ev_i, i, \textsf{post-job}_{hi} \cdot \textsf{apply}_i, \textsf{post-job}_{hi} \cdot \textsf{apply}_i \cdot \textsf{inform-outcome}_{ev_i})$$

$$c_3 : \mathsf{C}(i, ev_i, \textsf{make-offer}_{ev_i}, \textsf{make-offer}_{ev_i} \cdot (\textsf{response-yes}_i \vee \textsf{response-no}_i))$$

$$c_4 : \mathsf{C}(hi, ev_i, \textsf{post-job}_{hi} \cdot (\textsf{accepted}_{ev_i} \vee \textsf{timeout\_3months}_{hi}), \textsf{post-job}_{hi} \cdot \textsf{hiring}_{hi})$$

Where:

$$\begin{aligned}
\textsf{evaluate-candidate}_{ev_i} \equiv\ & \textsf{position-filled}_{hi} \cdot \textsf{msg-position-closed}_{ev_i} \vee \\
& (\textsf{screen-interview}_{ev_i} \cdot (\textsf{msg-rejection-notice}_{ev_i} \vee \textsf{make-offer}_{ev_i} \cdot \\
& (\textsf{response-yes}_i \cdot \textsf{accepted}_{ev_i} \vee \textsf{response-no}_i \cdot \textsf{offer-rejected}_{ev_i})))
\end{aligned}$$

$$\textsf{inform-outcome}_{ev_i} \equiv (\textsf{msg-position-closed}_{ev_i} \vee \textsf{msg-rejection-notice}_{ev_i} \vee \textsf{make-offer}_{ev_i})$$

$$\textsf{hiring}_{hi} \equiv (\textsf{accepted}_{ev_i} \cdot \textsf{position-filled}_{hi}) \vee (\textsf{timeout\_3months}_{hi} \cdot \textsf{position-abandoned}_{hi})$$

Fig. 9 The set of commitments included in the normative layer for the *Hiring Process* scenario.

a form of objective coordination. In this section we discuss how such a conceptualization finds a practical implementation by exploiting agent technology.

In general, we are prone to think of processes as procedures, and as such, we do not consider a process as an autonomous entity since every action and decision is already specified in the procedure itself. However, at a closer look, we can recognize that a process, which encodes an algorithm aimed at achieving some objective, is indeed autonomous. It has its own control flow, that is only influenced by the data relevant for the completion of the process itself. A process can therefore "select" the data that are to be used to produce a result in the sense that it implicitly "ignores" data that are not relevant nor well-formed. A process "decides" which data are relevant based on its internal state, represented by its private variables, and operates in an environment consisting of its inputs and outputs.

This characterization of a process closely resembles that of *agent* [39], given by Artificial Intelligence (AI). An agent is, by definition, *autonomous* and *situated* within an environment. Here, agents are "situated" in the sense that they perceive and manipulate the environment in which operate – as well as processes "perceive" and modify (relevant) data structures. Agents are "autonomous" in the sense that they implement a deliberative cycle, which gives them control of their internal state and behavior. Such a deliberative cycle naturally emerges as the control flow of the process "hidden" within the agent.

As already pointed out in [1], the notion of agent is an *abstraction of that of process*. This abstraction is justified by the fact that it enables a further form of software modularization in which both agent (process) and environment (data) are first-class elements of the software design [36]. It is worth noticing that such a modularization also recurs naturally in the conceptual architecture we have illustrated: the environment is populated by (business and coordination) artifacts that are shared by a number of principals. Thus, it is straightforward to map our proposal into a multiagent setting where principals are seen as agents.

4.1 Implementing the Hiring Process in JaCaMo+

In this paper we use the JaCaMo+ [3] platform for realizing the coordination architecture we propose. JaCaMo+ extends the well-known JaCaMo [10] platform with the social commitments as primitive programming elements. More precisely, JaCaMo is a conceptual model and programming platform that integrates agents, environments and organizations. It is built on top of Jason [11] for programming agents, CArtAgO [29] for programming environments, and $\mathcal{M}$OISE [22] for programming organizations. A MAS in JaCaMo consists of an agent organization, realized through $\mathcal{M}$OISE, organizing autonomous agents, programmed in Jason, working in a shared, artifact-based environment, programmed in CArtAgO.

For the purpose of this paper, it is sufficient to focus on the Jason and CArtAgO components. A Jason agent consists of a set of plans expressed as ECA-like rules (Event-Condition-Actions). 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:

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

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. In JaCaMo+, the triggering event can be a state change occurring in some active commitment.

CArtAgO is a possible realization of the A&A metamodel [36], where the environment is modeled as a set of computational resources, named *artifacts*. Artifacts can be manipulated by agents through a set of predefined operations made available by the artifacts themselves. Moreover, artifacts expose some *observable properties* that can be perceived by the agents, and that can trigger agents' plans. In particular, in JaCaMo+ the normative state, which contains the collection of the commitments that can be created along the execution and their states, is realized by means of a special class of artifacts (i.e., `CoordinationArtifact`).

Using JaCaMo+ as reference model for the implementation[1], we map the three principals' processes (hi, ev_i, and i) in Figure 6 onto Jason agents. Business artifacts positionBA and applicationBA and the normative state are instead naturally mapped into CArtAgO artifacts. 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 applicationBA on which the two agents operate. By exploiting the CArtAgO facility, the three artifacts are linked together, so the operations performed by the agents on the positionBA and applicationBA artifacts have effect on the hiringNormativeState, too, allowing the commitments to progress.

Implementing the Coordination Artifacts

The code snippet in Figure 10 sketches the implementation of the normative state for the *Hiring Process*. The artifact is an extension of `CoordinationArtifact` specifically provided by JaCaMo+ to support commitments. It is worth noting that the code is declarative in nature. The programmer, in fact, just specifies the commitments that can be created during the interaction (see method `initCommitments` in lines 7–22), and then specifies the operations that are possible upon this artifact (from line 25 on). In particular, antecedent and consequent conditions of commitments are specified in terms of `Fact` instances, and by using an extension of CArtAgO `LogicalExpression` in order to include the *before* operator of the precedent logic. Operations simply consists in the assertion of a specific `Fact` within the normative state, this corresponds to the occurrence of an event. Note that each operation is adorned with a `@LINK` annotation, this indicates that the operation is linked to some other operation on another artifact. Thus, whenever the business artifact operation is invoked, the linked operation on the normative artifact is invoked, too, enabling in this way the reporting of relevant information from the the business artifacts to the normative layer. Note that the progression of the commitments as events occur is captured by the underlying JaCaMo+ facilities (see line 5). This means that the management of the commitment lifecycles is not upon the programmer, making thus our implemen-

[1] The code of the whole system can be downloaded from `https://sourceforge.net/p/twocomm/code-svn/HEAD/tree/2COMMJaCaMoExamples/hiring/`.

```
public class HiringNormativeState extends CoordinationArtifact {
        ...
  public HiringNormativeState() {
        super();
        commitmentState = new AutomatedSocialStateSingleThreaded(this);
  }
 public void initCommitments(RoleId hirerId, RoleId evaluatorId, RoleId candidateId) throws ... {
        LogicalExpression ant;
        LogicalExpression cons;
        /*...definition of events occurring in antecedent and consequent of commitments... */
        Fact postJob = new Fact("postJob", hirerId.toString());
        Fact positionFilled = new Fact("positionFilled", hirerId.toString());
        Fact apply = new Fact("apply", candidateId.toString());
        ....
        /*...definition of antecedent and consequent expressions... */
        ant = new CompositeExpression(...);
        cons = new CompositeExpression(...);
        /*...commitments creation... */
        Commitment c1 = new Commitment(evaluatorId, hirerId, ant, cons);
        createCommitment(c1);
        ...
 }
 /*...artifact operations linked to (business) artifacts state changes... */
 @LINK
 public void postJob(RoleId hirerId) throws MissingOperandException {
        assertFact(new Fact("postJob", hirerId.toString()));
 }
 @LINK
 public void apply(RoleId candidateId) throws MissingOperandException {
        assertFact(new Fact("apply", candidateId.toString()));
 }
 @LINK
 public void screenInterview(RoleId evaluatorId) throws MissingOperandException {
        assertFact(new Fact("screenInterview", evaluatorId.toString()));
 }
 @LINK
 public void makeOffer(RoleId evaluatorId) throws MissingOperandException {
        assertFact(new Fact("makeOffer", evaluatorId.toString()));
 }
 @LINK
 public void updateFilled(RoleId hirerId) throws MissingOperandException {
        assertFact(new Fact("positionFilled", hirerId.toString()));
 }
 ...
}
```

Fig. 10 The outline of the `hiringNormativeState` artifact implementation of Figure 6.

tation a practical and simple tool to be used for the coordination of business processes.

Implementing the Agents

In order to implement the agents, it is possible to rely on a set of predefined patterns by exploiting some pragmatic rules. Telang *et al.* [35] studied the relationships between goals and commitments, capturing different ways in which the creation and evolution of the ones impacts on the creation and evolution of the others. The pragmatic rules include both rules from goals to commitments, and rules from commitments to goals.

Among those described in [35], the main rules are the following. The *entice rule* is used to create an interaction: when agent x seeks for support in achieving a goal p that it cannot obtain on its own, x makes an “offer” to another agent y by creating the commitment $c : \mathsf{C}(x, y, p, q)$. The intuition, here, is that y can achieve p, i.e., it is capable of performing a procedure to make p true, and it is interested in q which, on the other hand, is under the control of x. We can think of this commitment as a promise made by x to y that if y brings about p then x will provide y with q.

The creation of commitment c, thus, stimulates a form of cooperation between the two agents. The progression of such an interaction is driven by other practical rules. The *deliver rule* takes the point of view of agent y. If y, observing commitment c, estimates that q is an adequate achievement in exchange of the service p it can provide x with, it will engage the commitment by bringing about p. The deliver rule, thus, has the aim of making the commitment progress from conditional to detached, in this way the debtor agent x will be pushed bring about q.

The *discharge rule*, instead, helps the debtor agent selecting its next goals. The rule states that, if a commitment $c : \mathsf{C}(x, y, p, q)$ is detached, then agent x has to include q among its goals. In other terms, x is now committed to bring about q lest the violation of commitment c.

```
1 +!post-job <- postJob[artifact_id(positionBA)].
2 +cc($hi$, $ev_i$, post-job · (accepted ∨ timeout_3months$_{hi}$), post-job · hiring, DETACHED)[artifact_id(normativeState)]
3           : positionStatus(POSITION_OPEN)
4           <- updatePosition[artifact_id(positionBA)].
```

Listing 1 Hirer *hi*.

```
1 +cc($ev_i$, $hi$, post-job · apply , apply · evaluate-candidate, DETACHED)
2           : positionStatus(POSITION_OPEN)
3           <- screenInterview[artifact_id(applicationBA)];
4              // ... Choice
5              !offerOrReject(Choice).
6 +!offerOrReject(Choice) : Choice == yes
7           <- makeOffer[artifact_id(applicationBA)].
8 +!offerOrReject(Choice) : Choice == no
9           <- rejectionNotice[artifact_id(applicationBA)].
10 +responseYes($candidate_i$)[artifact_id(normativeState)]
11          : cc($ev_i$, $hi$, post-job · apply, apply · evaluate-candidate, DETACHED)
12          <- offerAccepted[artifact_id(applicationBA)]
13 +responseNo($candidate_i$)[artifact_id(normativeState)]
14          : cc($ev_i$, $hi$, post-job · apply, apply · evaluate-candidate, DETACHED)
15          <- offerRejected[artifact_id(applicationBA)].
16 +cc($ev_i$, $hi$, post-job · apply , apply · evaluate-candidate, DETACHED)
17          : not positionStatus(POSITION_OPEN)
18          <- positionClosed[artifact_id(applicationBA)].
```

Listing 2 Evaluator ev_i.

```
1 +postJob($hi$)[artifact_id(normativeState)] <- apply[artifact_id(applicationBA)].
2 +cc($candidate_i$, $ev_i$, make-offer, response-yes ∨ response-no, DETACHED)
3           <- // ... Choice
4              !response(Choice).
5 +!response(Choice) : Choice == yes
6           <- responseYes[artifact_id(applicationBA)].
7 +!response(Choice) : Choice == no
8           <- responseNo[artifact_id(applicationBA)].
```

Listing 3 Candidate *i*.

A specification of the commitments involved in an interaction is a means that helps programming the involved agents. For instance, in the Hiring Process, the designer of the evaluator process will use the entice rule to offer the hirer process the service of candidate evaluation provided that a job position is posted. In addition, the designer will use the discharge rule to carry out the evaluation when a position is actually posted. Similarly, the designer of the hirer process will use the deliver rule to exploit the evaluator's offer.

Listing 1 shows the body of the hirer process. The first rule, at line 1, means that when the hirer has the goal of posting a job position, it performs operation postJob on artifact positionBA. The second rule, at line 3, is instead originated following the *discharge rule* pattern: the hirer, in fact, has to properly react as soon as the commitment c_4 gets *detached*, and hence *hi* update the position status depending on the events that have satisfied the antecedent by performing operation updatePosition upon postJob.

The evaluator code in Listing 2 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 c_1 by interviewing the candidate, this corresponds to operation screenInterview performed upon business artifact applicationBA that is shared between the candidate and the evaluator. After this operation, the evaluator comes up with a Choice, either accept the candidate or reject it. This choice will, thus, active a proper behavior, and hence the corresponding operation on applicationBA, see the rules at lines 7 - 10. Rules at lines 12 - 17 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 applicationBA so as to progress in the evaluation process and, then, discharge its commitment. The rule starting at line 19 captures the situation in which the evaluator is held to inform the candidate as soon as the position gets filled. Interestingly, no specific rule is requested for treating commitment c_2 because whenever such a commitment gets detached, the evaluator satisfies it by satisfying c_1. However, at a normative layer, commitment c_2 is fundamental to detect the misbehavior of the evaluator towards the candidate.

Finally, Listing 3 sketches the pseudocode of a candidate, which applies for a position when a job is posted (see operation apply performed upon applicationBA), and, then, reacts to the detachment of commitment c_3 by answering either “yes” or “no” to an offer.

5 Conclusions

In this paper we have shown how business artifacts can be turned into coordination means in their own right, and that to this aim agent-based technology provides an adequate support. Specifically, social commitments, whose nature is to be “shared information” built on top of observable behavior, provide a normative interpretation layer to the state changes occurring to business artifacts.

Coordination in business applications is usually modeled via choreographies (see e.g., the recent proposal in [19]), even when processes and artifacts are addressed in a declarative way. For instance, besides the already cited BALSA methodology, which relies on event condition action rules, the GSM model [23] 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) [26], which is an extension and refinement of GSM.

Comparing our proposal with the above approaches, however, we can highlight many advantages, the first and most relevant one residing in the role played by a business artifact: in [9,23,26] a business artifact is just a piece of data, or as pointed out in [25], *the basis for factorization of knowledge* that enables business operations. In our proposal, business artifacts, through coordination artifacts, become the media through which interaction happens.

This idea has a software engineering solid foundation. In his survey on Concurrent Object-Oriented Languages (COOLs), Philippsen [28] highlights the importance of a *locality principle for class correctness*, and advocates that a way to achieve it is to realize a form of *coordination on the side of the callee.* Namely, the coordination is implemented in the class that is accessed concurrently. Moreover, Philippsen advocates that desirable properties of coordination code are *isolation* and *separability.* Isolation means that the code for coordination is isolated from the code that implements class functionality. Separability means that portion of the code for the coordination can be refined while other portions are reused. These two properties promote the modularity and reuse of code. Our proposal falls entirely in such a coordination model. The coordination is in fact implemented solely inside the artifact itself (i.e. the class called by the interacting parties). Consequently, coordination correctness can be assessed *locally.* Related to this, our proposal induces a form of *objective coordination* [27], where coordination is addressed outside the interacting agents. Objective coordination enables a clear separation between the implementation of the business logic and of the coordination logic. In our proposal, we meet this property by adding coordination artifacts. In this view, *data themselves become a coordination media.* Communication becomes *generative* [13], in the sense that agents communicate by generating data in the dataspace, and these data are available to any agent having access to the dataspace. This vision is in contrast to the message passing paradigm, at the basis of choreographies, where communication is only enabled between processes sharing the same channel.

Our proposal contributes also to research on MAS. Works on coordination protocols from the research area on MAS mainly focus on the sequence of messages that can be exchanged between two communicating agents, and disregard the information conveyed by these messages. Recent approaches, like HAPN [37] and BSPL [34], have started to consider also the information dimension. HAPN is formally based on automata, whose nodes represent states of the interaction and guarded transitions between nodes represent the messages that can be exchanged. A similar approach is followed by BSPL, where the information flow is decomposed in a number of “simple protocols”, each defining the schema of the messages that can be exchanged together with their parameters. BSPL protocols can be verified against properties like liveness or safety.

These approaches, however, show weaknesses in information handling. In HAPN, the protocol (e.g. transition guards) can refer to information which is not carried by the messages, but rather is maintained in an external information system –not an integral part of HAPN. This hinders interaction verification. BSPL, on its side, assumes a distributed view of information, i.e. each participant has its own knowledge base, which is modified as the interaction progresses. The problem is that each participant has just a local view of the information lifecycle, and this does not allow agents to create expectations about the behaviors of other participants as a consequence of the messages it sends. The approach we propose, by providing a comprehensive view that accounts for all such dimensions as data, information, and coordination overcomes these limitations.

This work can be extended along many lines of research. First of all, an explicit normative layer paves the way to formal verification. For instance, the notion of *computational accountability* [6,5] is gaining momentum, and it would be interesting to study how to support accountability by design inside information systems, based on the proposal we have explained. Especially in those cases where a complex task can only be accomplished by decomposing it into a number of concurrent processes (as in the hiring scenario discussed in this paper), the notion of accountability gains importance in modeling the mutual bindings among all the participants. As discussed in [7], it is convenient to adopt a goal-oriented perspective. Broadly speaking, goals capture the tasks assigned to the agents. Accountability relationships among agents, instead, specify the mutual expectations, and bind the activities of an agent as functional to the achievement of some other agent's goal.

An issue that the proposal does not currently tackle is data integration (see [24]): a coordination artifact is directly affected by modifications that possibly occur in many business artifacts. An alternative that deserves investigation is to sit the coordination artifacts onto a global view of the information in the system, that is obtained through data integration techniques, applied to the business artifacts seen as data sources. This approach would implement a separation of concerns between source integration and coordination, enhancing the modularity of the system.

To conclude, we mention RAW-SYS [18], which enriches the prescriptive process model with data-awareness. Although RAW-SYS looks similar to a coordination artifact, the objectives of the two models are quite different. RAW-SYS is essentially a framework for verifying business processes taking into account both the control- and the data-flows. A coordination artifact, instead, aims at coordinating autonomous agents.

Acknowledgements The authors would like to thank Olivier Boissier (Laboratoire Hubert Curien UMR CNRS 5516, Institut Henri Fayol, MINES Saint-Etienne, Saint-Etienne, France) and Jomi Fred Hübner (Department of Automation and Systems Engineering, Federal University of Santa Catarina, Florianópolis, Brasil) for the interesting discussions, and for having suppported the improvement of the draft. Special thanks to Stefano Tedeschi who supported the implementation. This work was partially supported by the *Accountable Trustworthy Organizations and Systems (AThOS)* project, funded by Università degli Studi di Torino and Compagnia di San Paolo (CSP 2014).

References

1. Baldoni, M., Baroglio, C., Calvanese, D., Micalizio, R., Montali, M.: Towards Data- and Norm-aware Multiagent Systems. In: Post-Proc. of the 4th International Workshop on Engineering Multi-Agent Systems, EMAS 2016, Revised Selected and Invited Papers, no. 10093 in LNAI, pp. 22–38 (2016)
2. Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Objective coordination with business artifacts and social engagements. In: Proc. of First Workshop on Business Process Innovations with Artificial Intelligence, BPAI (2017)
3. Baldoni, M., Baroglio, C., Capuzzimati, F., Micalizio, R.: Commitment-based Agent Interaction in JaCaMo+. Fundamenta Informaticae **157**, 1–33 (2018)
4. Baldoni, M., Baroglio, C., Marengo, E., Patti, V., Capuzzimati, F.: Engineering commitment-based business protocols with the 2CL methodology. JAAMAS **28**(4), 519–557 (2014). DOI 10.1007/s10458-013-9233-1. URL `http://dx.doi.org/10.1007/s10458-013-9233-1`
5. Baldoni, M., Baroglio, C., May, K.M., Micalizio, R., Tedeschi, S.: Computational Accountability. In: F. Chesani, P. Mello, M. Milano (eds.) Deep Understanding and Reasoning: A challenge for Next-generation Intelligent Agents, URANIA 2016, vol. 1802, pp. 56–62. CEUR, Workshop Proceedings, Genoa, Italy (2016)
6. Baldoni, M., Baroglio, C., May, K.M., Micalizio, R., Tedeschi, S.: ADOPT JaCaMo: Accountability-driven organization programming technique for JaCaMo. In: PRIMA 2017 - 20th Int. Conf., Proceedings, *LNCS*, vol. 10621. Springer (2017)
7. Baldoni, M., Baroglio, C., Micalizio, R.: Goal distribution in business process models. In: G. Chiara, B. Magnini, A. Passerini, P. Traverso (eds.) AI*IA 2018 - Advances in Artificial Intelligence - XVIIth International Conference of the Italian Association for Artificial Intelligence, Trento, Italy, November 20-23, 2018, Proceedings, *Lecture Notes in Computer Science*, vol. 11298, pp. 252–265. Springer (2018). DOI 10.1007/978-3-030-03840-3_19. URL `https://doi.org/10.1007/978-3-030-03840-3_19`
8. Bhattacharya, K., Caswell, N.S., Kumaran, S., Nigam, A., Wu, F.Y.: Artifact-centered operational modeling: Lessons from customer engagements. IBM Systems Journal **46**(4), 703–721 (2007)
9. Bhattacharya, K., Hull, R., Su, J.: A data-centric design methodology for business processes, pp. 503–531. Handbook of Research on Business Process Modeling. IGI Publishing (2009)
10. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. Science of Computer Programming **78**(6), 747 – 761 (2013). DOI http://dx.doi.org/10.1016/j.scico.2011.10.004. URL `http://www.sciencedirect.com/science/article/pii/S016764231100181X`
11. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons (2007)
12. de Brito, M., Hübner, J.F., Boissier, O.: Situated artificial institutions: stability, consistency, and flexibility in the regulation of agent societies. Autonomous Agents and Multi-Agent Systems **32**(2), 219–251 (2018)
13. Busi, N., Ciancarini, P., Gorrieri, R., Zavattaro, G.: Coordination Models: A Guided Tour, pp. 6–24. Springer Berlin Heidelberg (2001)
14. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data-aware process analysis: a database theory perspective. In: R. Hull, W. Fan (eds.) Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013, pp. 1–12. ACM (2013)
15. Castelfranchi, C.: Commitments: From individual intentions to groups and organizations. In: V.R. Lesser, L. Gasser (eds.) Proceedings of the 1st International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA, pp. 41–48. The MIT Press (1995)
16. Chopra, A.K.: Commitment Alignment: Semantics, Patterns, and Decision Procedures for Distributed Computing.

Ph.D. thesis, North Carolina State University, Raleigh, NC (2009)

17. Cohn, D., Hull, R.: Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. IEEE Data Eng. Bull. **32**(3), 3–9 (2009)
18. De Masellis, R., Di Francescomarino, C., Ghidini, C., Montali, M., Tessaris, S.: Add data into business process verification: Bridging the gap between theory and practice. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA., pp. 1091–1099 (2017)
19. Decker, G., Weske, M.: Interaction-centric modeling of process choreographies. Information Systems **36**(2), 292–312 (2011)
20. Desai, N., Chopra, A.K., Singh, M.P.: Amoeba: A methodology for modeling and evolving cross-organizational business processes. ACM Trans. Softw. Eng. Methodol. **19**(2) (2009). DOI 10.1145/1571629.1571632. URL http://doi.acm.org/10.1145/1571629.1571632
21. Dumas, M.: On the convergence of data and process engineering. In: Advances in Databases and Information Systems - 15th International Conference, ADBIS. Proceedings, *Lecture Notes in Computer Science*, vol. 6909, pp. 19–26. Springer (2011)
22. Hubner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems using the MOISE+ model: Programming issues at the system and agent levels. Int. J. Agent-Oriented Softw. Eng. **1**(3/4), 370–395 (2007). DOI 10.1504/IJAOSE.2007.016266. URL http://dx.doi.org/10.1504/IJAOSE.2007.016266
23. 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)
24. Lenzerini, M.: Data integration: A theoretical perspective. In: L. Popa, S. Abiteboul, P.G. Kolaitis (eds.) Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA, pp. 233–246. ACM (2002)
25. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal **42**(3), 428 – 445 (2003)
26. (OMG), O.M.G.: 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)
27. Omicini, A., Ossowski, S.: Objective versus subjective coordination in the engineering of agent systems. In: AgentLink, *Lecture Notes in Computer Science*, vol. 2586, pp. 179–202. Springer (2003)
28. Philippsen, M.: A survey of concurrent object-oriented languages. Concurrency - Practice and Experience **12**(10), 917–980 (2000)
29. Ricci, A., Piunti, M., Viroli, M., Omicini, A.: Environment Programming in CArtAgO, pp. 259–288. Springer US, Boston, MA (2009)
30. Schumacher, M.: Objective Coordination in Multi-agent System Engineering: Design and Implementation. Springer-Verlag, Berlin, Heidelberg (2001)
31. Silver, B.: BPMN Method and Style, with BPMN Implementer's Guide, second edn. Cody-Cassidy Press, Aptos, CA, USA (2012)
32. Singh, M.P.: An ontology for commitments in multiagent systems. Artificial Intelligence and 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 and Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings, pp. 907–914. ACM (2003)
34. Singh, M.P.: Information-driven interaction-oriented programming: BSPL, the blindingly simple protocol language. In: 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 491–498 (2011)
35. Telang, P.R., Yorke-Smith, N., Singh, M.P.: Relating Goal and Commitment Semantics. In: Proc. of ProMAS, *LNCS*, vol. 7212, pp. 22–37. Springer (2012)
36. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. JAAMAS **14**(1), 5–30 (2007)
37. Winikoff, M., Yadav, N., Padgham, L.: A new Hierarchical Agent Protocol Notation. Autonomous Agents and Multi-Agent Systems **32**(1), 59–133 (2018)
38. 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)
39. Wooldridge, M.J.: Introduction to multiagent systems. Wiley (2002)
40. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Trans. Softw. Eng. Methodol. **12**(3), 317–370 (2003). DOI 10.1145/958961.958963. URL http://doi.acm.org/10.1145/958961.958963