Modeling Process Interactions with Coordination Processes

Sebastian Steinau, Kevin Andrews, and Manfred Reichert

Institute of Databases and Information Systems, Ulm University, Germany
{sebastian.steinau,kevin.andrews,manfred.reichert}@uni-ulm.de

Abstract. With the rise of data-centric process management paradigms, small and interdependent processes, such as artifacts or object lifecycles, form a business process by interacting with each other. To arrive at a meaningful overall business process, these process interactions must be coordinated. One challenge is the proper consideration of one-to-many and many-to-many relations between interacting processes. Other challenges arise from the flexible, concurrent execution of the processes. Relational process structures and semantic relationships have been proposed for tackling these individual challenges. This paper introduces *coordination processes*, which bring together both relational process structures and semantic relationships, leveraging their features to enable proper coordination support for interdependent, concurrently running processes. Coordination processes contribute an abstracted and concise model for coordinating the highly complex interactions of interrelated processes.

Keywords: Process interactions, semantic relationships, many-to-many relationships, relational process structure, coordination process

1 Introduction

In enterprises, different entities need to collaborate to reach business objectives. The processes used to reach these objectives are not entirely executed in isolation, but have relations and are therefore interdependent. In particular, processes may depend on the execution status of several other processes, i.e., process dependencies may involve one-to-many or many-to-many-relationships. Corresponding interdependencies must be taken into account for the proper coordination of these concurrently executed processes. The proper coordination includes the challenge of coordinating multiple process instances, whose exact quantity is unknown at design-time and which may have different kinds of complex relationships with other process instances. Furthermore, the concurrent execution of processes may be asynchronous, i.e., a process depending on another process may only be synchronized at certain points in time. Finally, any coordination mechanism should impact the execution of the involved process instances as little as possible.

For dealing with the interdependencies between processes in one-to-many relationships, basic coordination patterns have been identified [15]. These patterns are denoted as *semantic relationships* and may be used to describe complex

coordination constraints among multiple process instances. As a prerequisite, semantic relationships require precise knowledge about which process instances are related to which other process instances at run-time. Furthermore, dynamic changes to the relations of process instances, i.e., the creation or deletion of process instances, must be tracked. A solution is the *relational process structure* [14]. While semantic relationships and the relational process structure each solve a part of the problem of process coordination, an overall concept bringing together both parts is still missing. Such a concept requires the specification of semantic relationships at design-time as well as the consideration of dynamic changes to process relations and the concurrent execution of process instances at run-time.

This paper presents *coordination processes*, which leverage both semantic relationships and the advantages of the relational process structure to provide a comprehensive coordination of interrelated process instances. Coordination processes not only support the concise specification of semantic relationships, but additionally allow for the appropriate semantic relationship to be automatically derived based on the relational process structure. Semantic relationships may be combined to express more complex constraints for process coordination. Furthermore, coordination processes take asynchronous execution of the coordinated processes into account by design. A coordination process interferes only when necessary, at certain points during the execution of a process instance, impacting its execution as little as possible. The concept of coordination processes originates from the object-aware approach to process management, where the coordination of the lifecycle processes of objects constitutes an integral part [9]. This paper contributes the support of *many-to-many relationships* in process coordination, which until now has been an open research challenge [6]. Further, the paper contributes a concise model and the ability to express sophisticated coordination constraints, allowing for the proper coordination of vast structures of interdependent processes in a comprehensive fashion.

The remainder of the paper is organized as follows: Section 2 recaps semantic relationships and relational process structures and characterizes their basic features. Section 3 introduces the concept of coordination processes. Additionally, it discusses the combination of different semantic relationships using ports and the customization of semantic relationships with expressions. Section 5 covers related work and discusses other approaches to process coordination. Finally, Section 6 concludes the paper with a summary and an outlook.

2 Semantic Relationships and Process Structures

Semantic Relationships and the relational process structure provide the fundamental concepts that enable the definition of coordination processes. This section provides a recap of relational process structures and semantic relationships. A running example from the human resource domain is used throughout the paper (cf. Example 1) to illustrate the concepts.

Example 1. (Recruitment Business Process)
In the context of recruitment, applicants may apply for job offers. The overall

process goal for a company is to determine who of the many applicants is best suited for the job. Applicants must write their application for a specific job offer and send it to the company. The company employees then evaluate each application by performing reviews. To reject an application or proceed with the application, a sufficient number of reviews need to be performed, e.g., the majority of reviews determines whether or not an application is rejected. If the majority of reviews are in favor of the application, the applicant is invited for one or more interviews, after which he may be hired or ultimately rejected. In the meantime, more applications may have been sent in, for which reviews are required, i.e., the evaluation of different applications may be handled concurrently, as well as the conduction of interviews. In particular, when an applicant is hired for the job offer, all other applicants are rejected.

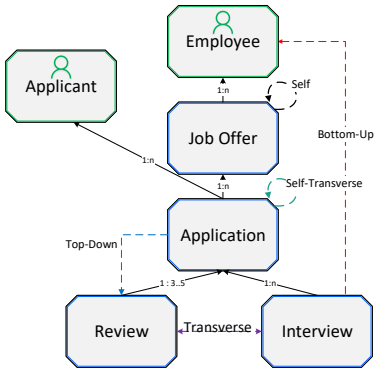

Fig. 1. Relational process structure and examples of semantic relationships

Example 1 describes many individual processes that are related to each other. At design-time, a *relational process structure* captures these processes and their relations [14]. A relation between processes indicates a dependency; on one hand, this explicit capturing of relations allows using these relations for various other purposes, such as specifying message exchanges between two related processes. On the other, it enables the detailed monitoring of process relations at run-time. The detailed knowledge about which process instances are related to which other process instances is crucial for proper process coordination. Figure 1 shows a relational process structure at design-time the processes of Example 1. In detail, the processes are *Job Offer*, *Application*, *Review*, and *Interviews*, whereas *Employee* and *Applicant* represent users, indicating responsibilities for creating and executing other processes.

Example 2. A *Job Offer* may be related to one or more *Applications*, which may have one or more relations to *Interviews*. In case of *Reviews*, the relation is restricted to at minimum three and at most five *Reviews* per *Application*. This cardinality restriction on a process relation is also enforced at run-time by the relational process structure. Furthermore, *Interview* and *Job Offer* are not directly related, but *transitively* via a path of relations. Allowing transitive relations allows for more expressiveness in the coordination of processes. At run-time, the relational process structure tracks the creation and deletion of process instances and their relations. Consequently, a relational process structure is able to give always up-to-date information about which process instances are related with each other. Semantic relationships leverage this capability of relational process structures to specify dependencies between processes and enforce them at run-time.

Table 1. Overview over semantic relationships

Name	Description of the semantic relationship
Top-Down	The execution of one or more lower-level processes depends on the execution status of one common higher-level process.
Bottom-Up	The execution of one higher-level process depends on the execution status of one or more lower-level processes of the same type.
Transverse	The execution of one or more processes is dependent on the execution status of one or more processes of different type. Both types of processes have a common higher-level process.
Self	The execution of a process depends upon the completion of a previous step of the same process.
Self-Transverse	The execution of a process depends on the execution process of other processes of the same type. All processes have a common higher-level process.

Semantic relationships may be used to model *coordination constraints* [15]. A coordination constraint is a formal or informal statement describing one or more conditions or dependencies that exist between processes. For example, statement "An application may only be accepted if three or more reviews are positive" is a coordination constraint. A coordination constraint must be expressed in terms of semantic relationships for the use in a coordination process. For a proper representation of coordination constraints, the combination of different semantic relationships might be necessary. A semantic relationship describes a recurring semantic pattern inherent in the coordination of processes in a one-to-many or many-to-many relationship (cf. Table 1). As one example of a pattern, several process instances may depend on the execution of one other process instance. A semantic relationship may only be established between processes if a path of relations in the relational process structure, i.e., a dependency, exists between these processes. Figure 1 shows examples of semantic relationships between different processes. In this context, the terms *lower-level* and *higher-level* refer to the fact that the relations are directed (cf. Figure 1). Process A is denoted as higher-level process in respect to a reference process B if there is a directed relation from B to A. Analogously, there may be many source processes C_i denoted as lower-level processes in respect to a reference process D. This terminology applies with transitive relations as well.

The execution status referred to in Table 1 is represented by a state-based view of the process [15]. Thereby, the process to be coordinated is abstracted and partitioned into different states that provide significant meaning for process coordination. Furthermore, as the state-based view abstracts from the underlying process language, any language might be used to model the process. For example, an *Application* has states *Sent* and *Checked*, which are important milestones for its coordination. An *active state* represents the current execution status. At run-time, based on the execution status, semantic relationships possess a logical value that indicates whether the represented condition is currently satisfied and, therefore, whether the execution of processes may progress or must halt. For example, a *Job Offer* has active state *Published*, and a top-down semantic

relationship has value *true* to indicate that now *Applications* may be created for the *Job Offer*. Apart from the state-based view, processes may provide access to data attributes for use in a coordination process.

However, a method to *effectively specify semantic relationships is still missing*. Furthermore, coordination constraints often need several semantic relationships to be expressed. As semantic relationships have a logical value to indicate whether or not they are satisfied, boolean operators are required to express more complicated coordination logic. Coordination processes combine the effective specification of semantic relationships with a graphical representation of boolean logic. A coordination process leverages the relational process structure and properties of semantic relationships to automatically derive the appropriate semantic relationship between two processes at design-time.

3 Coordination Processes

Coordination processes are a generic concept for coordinating processes by expressing coordination constraints with the help of semantic relationships, which are then enforced at run-time. The concept allows specifying sophisticated coordination constraints for vast structures of interrelated process instances with an expressive, high-level graphical notation using a minimum amount of modeling elements. The modeling elements are the *coordination step*, the *coordination transition,* and the *port*. Coordination processes follow a type-instance schema, where types (denoted ^{T}) represent design-time entities and instances (^{I}) run-time entities. Consequently, an instance is created by instantiating a type. The dot (.) represents the access operator.

Definition 1. *(Coordination Process Type)*
A coordination process type c^T *has the form* $(\omega^T, B^T, \Delta^T)$ *where*

- ω^T *is the process type to which the coordination process type* c^T *belongs.*
- B^T *is a set of coordination step types* β^T *(cf. Definition 3).*
- Δ^T *is a set of coordination transition types* δ^T *(cf. Definition 4).*

The *coordinating process type* ω^T determines the overall context of the coordination process, e.g., it determines the start and end coordination steps of the coordination process and which processes may be coordinated.

Definition 2. *(Process Type)*
A process type ω^T *has the form* (n, Σ^T, c^T) *where*

- n *is the name of the object type.*
- Σ^T *is a set of state types* σ^T*, representing the state-based view of* ω^T*.*
- c^T *is an optional coordination process type (cf. Definition 1). Default is* $\perp$*.*

For handling the complexities of dozens or hundreds of interrelated processes at design-time, abstraction in a coordination process is crucial. As a part of this effort, process types are represented with a state-based view, which abstracts

from process details and only exposes properties which are useful for process coordination.

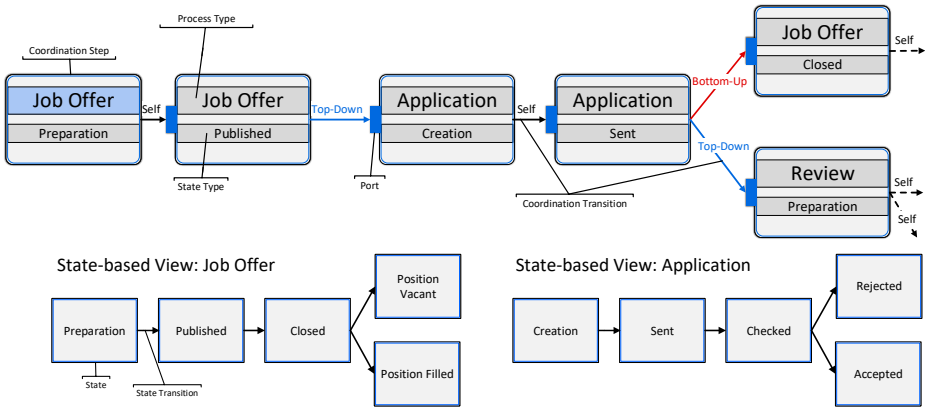

Fig. 2. Coordination Process Model and State-based Views, Part I

3.1 Coordination Steps and Coordination Transitions

Coordination processes are represented as a directed graph that consists of *coordination steps*, *coordination transitions* and *ports*. Figure 2 shows a part of the coordination process for Example 1 with *Job Offer* as the coordinating process type. Coordination steps are the vertices of the graph referring to a process type as well as to one of its states, e.g. *Job Offer* and state *Published*. For the sake of convenience, a coordination step is addressed with referenced process type and state in the form of *ProcessType*:*State*, e.g. *Job Offer*:*Published*. A formal definition for coordination steps is presented in Definition 3.

Definition 3. *(Coordination Step Type)*
A coordination step type β^T *has the form* $(c^T, \omega^T, \sigma^T, \Delta^T_{out}, H^T)$ *where*

- c^T *is the coordination process type (cf. Definition 1).*
- ω^T *is a reference to a process type.*
- σ^T *is a reference to a state type belonging to* ω^T*, i.e.,* $\sigma^T \in \omega^T.\Sigma^T$ *.*
- Δ^T_{out} *is a set of outgoing coordination transition types* δ^T *(cf. Definition 4).*
- H^T *is a set of port types* η^T *(cf. Definition 5).*

A coordination step type represents a collection of process instances of type ω^T at run-time. As such, a coordination step type provides a succinct and abstract way to represent *multiple process instances at run-time*, thus constraining much of the complexity of interdependent multiple process instances to the run-time instead of the design-time.

A coordination transition is a directed edge that connects a *source coordination step* with a *target coordination step* (cf. Figure 2). Both source and target coordination step reference a process type of the relational process structure. By creating a coordination transition between source and target step, a

semantic relationship is created as well. Conceptually, a semantic relationship is attached to a coordination transition. With the relations from the relational process structure and the definitions of semantic relationships (cf. Table 1), it can be automatically derived which semantic relationship is established between the process types referenced by the two coordination steps.

Example 3. (Top-Down and Bottom-Up Semantic Relationships)
Connecting *Job Offer*:*Published* with *Application*:*Sent* constitutes a top-down relationship (cf. Figure 2). The sequence in which the steps occur is important for determining the type of semantic relationship. Connecting *Application*:*Sent* with *Job Offer*:*Closed*, a bottom-up semantic relationship is established instead, as *Application* is a lower-level process type of *Job Offer*.

A formal definition for coordination transitions can be found in Definition 4.

Definition 4. *(Coordination Transition Type)*
A coordination transition type δ^T *has the form* $(\beta^T_{src}, \eta^T_{tar}, s^T)$ *where*

- β^T_{source} *is the source coordination step type (cf. Definition 3).*
- η^T_{target} *is the target port type (cf. Definition 5).*
- s^T *is a semantic relationship between* $\beta^T_{src}.\omega^T$ *and* $\eta^T_{tar}.\beta^T.\omega^T$.

For the sake of convenience, the terminology of source or target coordination step of a coordination transition applies for the corresponding semantic relationships as well. The strict distinction between coordination transition and semantic relationship is crucial at run-time and is therefore reflected in the design-time model. For establishing a semantic relationship between two processes, the state σ^T of any coordination step is not relevant, only the process types are relevant. However, states becomes crucial for the actual representation and enforcement of coordination constraints at run-time. Depending on the activation of states at run-time, semantic relationships become enabled or disabled.

Example 4. (Top-down Semantics)
Figure 2 depicts coordination steps *Job Offer*:*Published* and *Application*:*Creation*. The top-down semantic relationship between these coordination steps enforces that a *Job Offer* must reach state *Published* before any application may be created (i.e., *Creation* is the start state of an *Application*). Once a particular *Job Offer* reaches state *Published* and the state becomes active, the top-down semantic relationship becomes enabled and subsequently allows creating any number of *Applications* for the *Job Offer*, at different points in time.

Coordination processes only permit or prohibit the activation of states. The actual activation is determined by the process itself, i.e., by its progress. It is therefore possible that a coordination process allows activating a state long before actually reaching this state of the process. On the other side, a coordination process may halt process execution until the specified coordination constraints are fulfilled, i.e., the semantic relationships become enabled.

Enabling a semantic relationship requires that all predecessor semantic relationships in the coordination process have been enabled as well, e.g., enabling the

bottom-up semantic relationship of *Application*:*Sent* with *Job Offer*:*Closed* requires that the top-down semantic relationship *Job Offer*:*Published* with *Application*:*Sent* has already been enabled (cf. Figure 2). As a consequence, a coordination process graph must be acyclic and connected. If a coordination process contained a cycle, it would result in an immediate deadlock once a coordination step of the cycle is reached. Due to the cycle, its incoming semantic relationships never become enabled. Furthermore, the start and end coordination steps of a coordination process must reference the coordinating process type, i.e., $\beta^T.\omega^T = c^T.\omega^T$, ensuring its proper start and completion.

Semantic relationships are based on one-to-many relationships. This includes transitive relations, e.g., a semantic relationship may be established between *Job Offer* and *Interview*. If the processes are in a many-to-many (*m:n*) relationship and are in a (w.l.o.g.) top-down semantic relationship, a coordination process replicates the top-down semantic relationship m times at run-time, depending on the number m of higher-level processes. It is thereby established that each of the n lower-level processes depends on each of the m higher-level processes. In consequence, many-to-many-relations may be elegantly coordinated.

So far, just based on coordination steps and coordination transitions, only simple coordination constraints may be expressed, i.e., constraints that may be represented by a single semantic relationship. However, coordination constraints may require multiple semantic relationships to be properly represented. Additionally, the states of a process may be involved in several coordination constraints, requiring all of them to be fulfilled in order to become enabled. Coordination processes therefore incorporate the concept of *ports*, which allow combining multiple semantic relationships for a state to become active.

3.2 Ports

Coordination transitions do not target a coordination step directly, but instead target a port attached to a coordination step. Any coordination step must have one or more ports (with the exception of the start coordination step).

Definition 5. *(Port Type)*
A port type η^T *has the form* (β^T, Δ_{in}^T) *where*

- β^T *is the coordination step type to which this port type belongs.*
- Δ_{in}^T *is the set of all incoming coordination transitions* δ^T *(cf. Definition 4).*

Ports allow realizing different semantics for combining semantic relationships. Connecting multiple transitions to the same port corresponds to AND-semantics, i.e., all semantic relationships attached to the incoming transitions must be enabled for the port to become enabled as well. Enabling a port also enables the coordination step, allowing the state of the coordination step to become active. Generally, at least one port of a coordination step must be enabled for the coordination step to become enabled as well. Consequently, connecting transitions to different ports corresponds to OR-semantics.

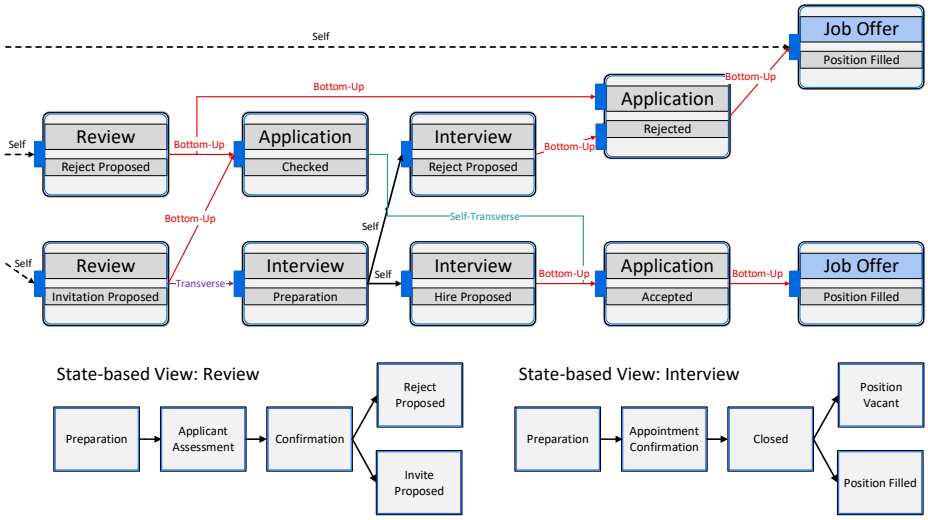

Fig. 3. Coordination Process Model and State-based Views, Part 2

Example 5. (Port AND Semantics)
A coordination constraint may state that for an application to be accepted, a sufficient number of interviews must propose a hire. Implicitly, a job may not be given to two different applicants; therefore, no other application must have already been accepted for the same job offer. To model this coordination constraint, coordination step *Application*:*Accepted* (cf. Figure 3) has one port with two incoming coordination transitions. Therefore, for an *Application* to be accepted, both conditions represented by the semantic relationships need to be fulfilled. The bottom-up semantic relationship outgoing from coordination step *Interview*:*Hire Proposed* requires a sufficient number of *Interviews* to have reached state *Hire Proposed* before an *Application* may be accepted. The process type and state combination required to enable the semantic relationship is determined by its source coordination step. The exact number of *Interviews* required to be in state *Hire Proposed* is a design choice.

The implicit condition is represented by a *self-transverse* semantic relationship, i.e., an *Application* depends on the execution status of other *Applications* in context of the same *Job Offer*. In this case, only exactly one *Application* may reach state *Accepted*, i.e., once an *Application* has been accepted, other *Applications* are prevented from reaching state *Accepted*. The AND-semantics of the target port require that both conditions are *true* at the same time so the *Application* may be accepted.

Example 6. (Port OR Semantics)
Rejecting an *Application* may be achieved in two different ways. First, the *Reviews* corresponding to the *Application* favor an immediate rejection. Second, during one or more *Interviews*, a rejection of the application is favored, and the *Application* is rejected then. In both cases, the corresponding semantic

relationship is bottom-up, but connects to two different ports of the coordination step *Application*:*Rejected* (cf. Figure 3). The OR-semantics, therefore, allows rejecting the *Application* in either case.

Both AND and OR semantics may be combined to express even more complex coordination constraints. Basically, connecting semantic relationships to ports allows building boolean formulas. When viewing semantic relationships as literals, ports are similar to clauses in a disjunctive normal form (DNF) of boolean logic. In summary, ports enhance the expressiveness of semantic relationships and coordination processes significantly. However, most coordination constraints may still not be adequately represented, as semantic relationships have only been used in their basic forms so far. Section 3.3 explores how semantic relationships can be configured to express sophisticated coordination constraints.

3.3 Configuring Semantic Relationships

All semantic relationships, except the self semantic relationships, provide configuration options to the process modeler [15]. This allows for fine-grained control over the basic semantics of the semantic relationship, increasing the degree to which complex coordination constraints may be expressed. A top-down semantic relationship must specify when it is no longer enabled, due to progression in the higher-level process (cf. Table 1). For example, a *Job Offer* may no longer accept new *Applications* after state *Closed* has become active. However, the exact means to achieve this are not specified by the top-down semantic relationship. Coordination processes rectify the situation by introducing a *state set*. A top-down semantic relationship becomes enabled once the state of the source coordination step, denoted as the *base state* of the top-down semantic relationship, becomes active. For example, reaching base state *Published* of a *Job Offer* enables the outgoing top-down semantic relationship (cf. Figure 2).

Consequently, the base state must automatically be part of the state set. As long as the currently active state of the higher-level process belongs to this state set, the top-down semantic relationship remains enabled. Successor states of the base state may also be added to the state set by the process modeler. This keeps the top-down semantic relationships enabled while the higher-level process progresses, as long as its active state belongs to the state set.

Example 7. (Top-Down Configuration)
Suppose that state *Closed*, a successor state of state *Published*, is added to the state set of the top-down semantic relationship. Then, new *Applications* may still be added even when the *Job Offer* is closed, i.e., is in state *Closed*. The top-down semantic relationship becomes disabled as soon as the *Job Offer* reaches either state *Position Filled* or state *Position Vacant*, i.e, *Applications* may no longer establish new relations to the *Job Offer*.

As opposed to top-down semantic relationships, bottom-up, transverse, and self-transverse semantic relationships can be configured by using expressions. Such an expression is denoted as a *coordination expression* and represents more

specialized conditions, in addition to the basic semantics of the respective semantic relationship. For example, the bottom-up relationship between *Interview*:*Hire Proposed* and *Application*:*Accepted* requires a sufficient number of *Interviews* in state *Hire Proposed*. With an expression, this condition can be specified formally and with a concrete number.

Example 8. (Expressions)
A process modeler may specify that at least two *Interviews* are necessary for a hire. This may be represented as $Count(Interview, Hire\ Proposed) \geq 2$, where *Count* is a function. In principle, the required expressions may be of arbitrary complexity, allowing for the full range of boolean and arithmetic functions, constants, and variables based on process data. In particular, negating the semantics of semantic relationships is possible.

In Example 8, the notation of the expression does not incorporate the given context for which this expression must be evaluated at run-time. In fact, *Count* must not be evaluated on a global level, i.e., counting every *Interview* of every *Applicant* for every *Job Offer*, but must be evaluated in context of a single *Application*. Otherwise, this would have undesired and even absurd side effects, such as that two positive interviews (for any two applicants) would allow additional applicants to be accepted as well. Therefore, it is essential that the context is reflected in the expression framework, while keeping the expressions simple. Often, it is desired that an expression framework shows high expressiveness to solve the particular problem at hand. However, high expressiveness usually comes with a number of drawbacks. Among these drawbacks, two may be considered as the most severe. First, high expressiveness is generally correlated with high complexity. This causes difficulties when specifying expressions, as substantial knowledge of the expression framework is necessary for modeling. Second, high complexity poses problems in the implementation of the expressions, requiring considerable efforts to support all possible expressions in all possible combinations. Thus, less used or more complex expressions are often not implemented due to time and resource constraints, limiting the use of the expression framework in practice. With the clear focus of expressions in semantic relationships, it becomes possible to reduce the complexity of the expressions.

Several models[1] that involve coordination processes have shown that a high percentage of the expressions used for configuring semantic relationships require the *counting* of process instances. The instances to be counted are represented by the source and target coordination steps. Furthermore, counting depends on the state of the process, e.g., it is important how many processes are in a particular active state. In other cases, it is important whether a particular state has been active, has not yet been active, or has been skipped due to the selection of alternative execution paths in the process. Due to the state-based views of the involved processes, the status of states is of particular concern to the coordination of processes at run-time. Therefore, it is beneficial to define specialized counting functions as part of the expression framework.

[1] A selection has been approved for publication, available at `https://bit.ly/2yo6GTe`

Definition 6. *(Coordination Expression Counting Functions)*
Let s^T be a semantic relationship and δ^T be a coordination transition. Let Ω^I_{source} be the process instances of type $\delta^T.\beta^T_{source}.\omega^T$ coordinated by s^T. Let Σ^T be the state type set of $\delta^T.\beta^T_{source}.\omega^T$. Then:

- $\#AllSource: \Omega^I_{source} \rightarrow \mathbb{N}_0$
  Determines the total number of process instances for s^T.
- $\#InSource: \Omega^I_{source} \times \Sigma^T \rightarrow \mathbb{N}_0$
  Determines the number of process instances of s^T where state σ^I of type $\sigma^T \in \Sigma^T$ is currently active.
- $\#BeforeSource: \Omega^I_{source} \times \Sigma^T \rightarrow \mathbb{N}_0$
  Determines the number of process instances of s^T where state σ^I of type $\sigma^T \in \Sigma^T$ has not yet been active, i.e., a predecessor state is active.
- $\#AfterSource: \Omega^I_{source} \times \Sigma^T \rightarrow \mathbb{N}_0$
  Determines the number of process instances of s^T where state σ^I of type $\sigma^T \in \Sigma^T$ has been active in the past, i.e., a successor state is active.
- $\#SkippedSource: \Omega^I_{source} \times \Sigma^T \rightarrow \mathbb{N}_0$
  Determines the number of process instances of s^T where state σ^I of type $\sigma^T \in \Sigma^T$ is not on the execution path of the process instances, i.e, a mutual exclusive state to σ^I is active.

Analogously, functions can be defined that count process instances of Ω^I_{target} with type $\delta^T.\eta^T_{target}.\beta^T.\omega^T$.

With these functions, expression $Count(Interview, Hire\ Proposed) \geq 2$ can be redefined, explicitly taking context, i.e., the respective semantic relationship, into account. Figure 4 shows an excerpt from the coordination process from Figures 2 and 3. The semantic relationships have been annotated with their respective coordination expressions. For the bottom-up semantic relationship, counting function $\#InSource$ has been used, as the source coordination step is $Interview$:$Hire\ Proposed$. Using the counting function $\#InSource$, in conjunction with the source or target coordination step and the respective semantic relationship, therefore clearly defines the context for evaluating the expression.

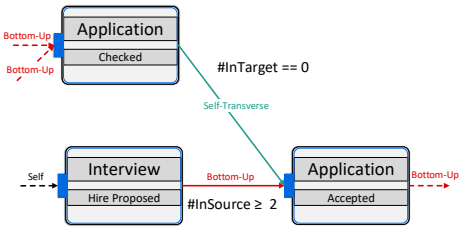

Fig. 4. Counting Functions Example

At run-time, as soon as two *Interviews* reach state *Hire Proposed*, the semantic relationship becomes enabled. For the self-transverse semantic relationship, the counterpart counting function $\#InTarget$ has been used, counting *Applications*. In line with the semantics of a self-transverse semantic relationship (cf. Table 1), an *Application* may only be accepted if no other *Application* has already been accepted. As at run-time initially no *Application* is accepted and thus, the coordination expression is *true*, which means the self-transverse semantic relationship becomes enabled once an

Application reaches state *Checked*. If an *Application* becomes accepted, the self-transverse semantic relationship is disabled due to the coordination expression $\#InTarget == 0$ no longer evaluating to *true*. Therefore, no more *Applications* may reach state *Accepted*.

In case a modeler has not specified a coordination expression, bottom-up, transverse, and self-transverse semantic relationships default to the expression $\#InSource = \#AllSource$, meaning the referenced state must be active in all process instances. Note that these functions are intended to facilitate frequently encountered use cases when specifying semantic relationships, the expression framework is not limited to using these functions. In previous work [9], expression were limited to counting source processes, severely limiting the expressiveness of semantic relationships. With the addition of the target coordination expressions (e.g., $\#InTarget$) and other types of expressions, which are not replicated here for the sake of brevity, a wider range of coordination constraint can be realized.

3.4 Operational Semantics of Coordination Processes

The concept of coordination processes not only comprises the modeling of process interactions, but includes *operational semantics* as well. The operational semantics defines the run-time behavior of coordination processes. The highly dynamic nature of the relational process structure at run-time and the frequent state changes of processes create a unique set of challenges for a coordination process at run-time. This requires a high flexibility to tackle these challenges on part of the coordination process. For example, as processes may execute concurrently, semantic relationships must cope with different processes reaching particular states at different points in time and in different order. Furthermore, creating and deleting processes or changing relations of interconnected processes all affect a coordination process, i.e., coordination constraints become fulfilled or are no longer fulfilled. The operational semantics must account for all eventualities to ensure a correct process execution of all involved process instances.

For this purpose, a coordination step type may be instantiated multiple times at run-time, each representing one process instance of the relational process structure. Consequently, semantic relationships are also instantiated multiple times at run-time, allowing for the coordination of different sets of processes independently and contextually. For example, *Application A* is related to two *Interviews* and *Application B* is related to none. For each *Application*, a semantic relationship is instantiated. As a consequence, coordination processes form complex, interconnected structures at run-time. Note that the run-time of coordination processes is too complex to be presented in this paper in its entirety and must therefore be reserved for future publications.

In summary, coordination processes are a powerful concept for coordinating processes in one-to-many and many-to-many relationships. They combine the capabilities of relational process structures and the semantic relationships to model coordination constraints and enforce these constraints at run-time. With ports and coordination expressions, coordination processes may be customized

to fit individual needs while retaining the basic semantics of the individual semantic relationships. Modeling is facilitated by the comparatively low number of modeling elements and the fact that semantic relationships may be derived automatically when connecting coordination steps with a coordination transition, which is possible due to the underlying relational process structure. With this, managing the challenges of multiple interrelated processes at run-time is possible. Furthermore, semantic relationships, as the cornerstone of coordination processes, allow reacting correctly to the changes during lifecycle execution. The relational process structure ensures that a coordination process has always up-to-date information on every process instance and its relations. Finally, a coordination process model is designed so that it is immediately executable upon instantiation, i.e., there is no distinction between functional and technical model.

4 Proof-of-Concept: Demonstrating the Feasibility of Coordination Processes

Object-aware process management [9] has been centered around the idea that objects with lifecycles and their interactions constitute a business process. As a data-centric paradigm, objects acquire data according to their lifecycle processes, i.e., the change in progress is data-driven. In particular, many process instances of a type may exist, which have interdependencies to other process instances. Consequently, coordination processes have been developed to steer these different interacting lifecycles in order to reach a meaningful overall business process. Objects, their lifecycles and coordination processes constitute the core of a business process management system prototype. This prototype is based on the object-aware approach and has been developed in the PHILharmonicFlows[2] project at Ulm University.

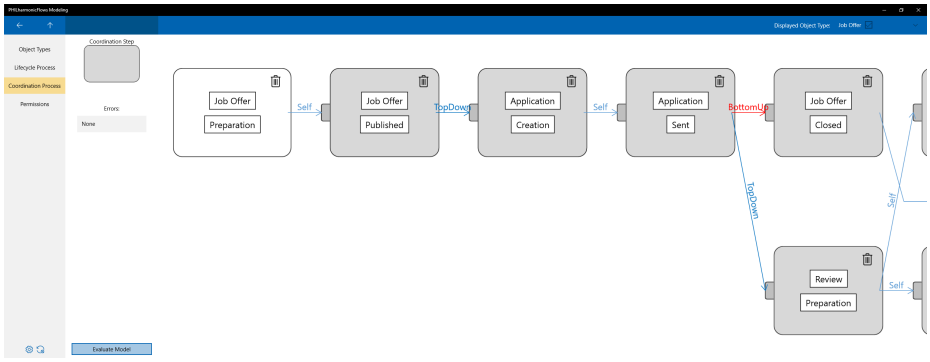

Fig. 5. PHILharmonicFlows Modeling Tool Showing a Coordination Process

The prototype comprises a tool for modeling data models with objects together with their lifecycle processes and relations to other objects, i.e., a relational process structure. The tool also supports the modeling of coordination

[2] For more details on the prototype visit `https://bit.ly/2KYvyT9`

processes as described in this paper. Figure 5 shows the coordination process of the running example (cf. Figure 2 and 3) modeled with the tool. The prototype additionally comprises a run-time environment, which is able to asynchronously execute both lifecycle and coordination processes with the required flexibility. The prototype uses a micro service architecture for high scalability and parallel, asynchronous execution of processes, as objects and their attached processes are uniquely suited to be distributed among such micro services. This raises further challenges for coordination processes when employed in large-scale, distributed relational process structures.

Furthermore, coordination processes were successfully used to model various processes, both real-world and exploratory examples. Some have been modeled in cooperation with industrial partners. All models comprise dozens of object types and multiple coordination processes[3]. The models showed that, in general, coordination processes are able to represent coordination constraints adequately. While modeling of coordination processes requires extensive knowledge of several concepts, e.g., semantic relationships, it is by far compensated by the built-in executability of the models.

5 Related Work

Coordination processes support various features rarely seen in other process coordination approaches, most notably the support of many-to-many relationships. This gives coordination processes a unique advantage. Table 2 shows a comparison between coordination processes and selected related work. Note that Table 2 compares approaches according to specific features and therefore does not represent an overall quality assessment of the individual approaches.

Table 2. Comparison of Process Coordination Approaches

	Artifact-centric (GSM)	Proclets	BPMN	Corepro	Coordination Processes
paradigm-agnostic					✔
explicit relations				✔	✔
transitive relations	(✔)				✔
many-to-many relations	(✔)				✔
process cardinality	(✔)	✔	(✔)		✔
message-based	(✔)	✔	✔	✔	

✔ : Supported (✔) : Indirectly supported

Artifact-centric process management [12] uses the Guard-Stage-Milestone (GSM) meta-model [7,8] for process modeling. Central to this approach is the *artifact*, which holds all process-relevant information. It may further interact with other artifacts. However, GSM does not provide dedicated coordination

[3] A selection has been approved for publication, available at `https://bit.ly/2yo6GTe`

mechanisms or explicit artifact relations, and therefore does not support any criterion of Table 2. Instead, GSM incorporates an arbitrary information model and a sophisticated expression framework that, in principle, allow fulfilling the comparison criteria with expressions and custom data. As a drawback, expressions might become very complex and explicitly need to be integrated into the process model. Therefore, model verification [1,2,4] constitutes an important aspect of artifact-centric process management. Further, [6] recognizes the need for supporting many-to-many relationships in artifact-centric choreographies.

For artifact-centric process models based on Finite-State-Machines (FSMs), [16] developed a message-based declarative artifact-centric choreography. This approach proposes the use of exactly one master artifact to coordinate all artifacts in a correlation graph. The approach explicitly considers the run-time presence of multiple instances. While the approach shows some similarities to coordination processes, the message-based coordination mechanism neither provides the run-time flexibility of semantic relationships nor the succinct model of a coordination process. Moreover, it is unclear if and how the findings translate from FSM-based to GSM-based artifacts.

Proclets [17] are lightweight processes with focus on process interactions. Proclets interact via messages called *Performatives.* Proclets allow specifying the cardinality for a message multicast, i.e., the number of Proclets that receive a performative. Proclets are capable of asynchronous and concurrent execution. However, relations between different Proclets are not considered. Proclets are defined using Petri nets, which are extended with ports that send and receive performatives. The concept of ports in the Proclet approach is fundamentally different than ports in coordination processes.

The coordination of large process structures with focus on the engineering domain is considered in [10,11]. The COREPRO approach explicitly considers process relations with one-to-many cardinality and dynamic changes at runtime, but transitive relations are not considered. In comparison to COREPRO, semantic relationships correspond, in principle, to external state transitions of a Lifecycle Coordination Model. However, the external state transitions do not take the semantics of the respective process interaction into account.

Regarding the activity-centric process modeling paradigm, several approaches enable a specific kind of coordination. For activity-centric processes, workflow patterns have been identified [18]. Several workflow patterns describe interactions between processes, which may be used for coordinating processes. The business process architecture approach [5] also identifies generic patterns to describe a coordination between processes. iBPM [3] enhances BPMN to support coordination of processes by modeling process interactions.

The BPMN standard [13] provided the choreography diagram explicitly dedicated to model the interactions between processes. Similar to coordination processes, choreography diagrams abstract from the coordinated processes and only display interactions themselves. The coordinated process types can be annotated with single-instance and multi-instance markers, showing very limited support

in restricting process cardinality. Similar to coordination processes, they are displayed on separate diagrams and posses few modeling elements.

Common to all these approaches, with the exception of artifacts, is the use of messages as a mechanism for coordination. While the exchange of messages allows for a detailed process coordination, all message flows have to be identified, the contents of the messages defined, and the recipients determined. This constitutes an enormous complexity when facing numerous processes that need to be coordinated, and in many cases, it impairs the flexible execution of the involved processes. Except Proclets, the modeling of coordination aspects is integrated into the actual process models, increasing the complexity to the process models. Coordination processes allow expressing complex interdependencies concisely using semantic relationships.

6 Summary and Outlook

A coordination process is an advanced concept for coordinating a collection of individual processes. It provides the superstructure to effectively employ relational process structures and semantic relationships. A coordination process itself is specified in a concise and comprehensive manner using coordination steps, coordination transitions and ports, abstracting from the complexity of coordinating a multitude of interrelated processes. Coordination processes allow for the automatic derivation of semantic relationships from connecting two coordination steps with a coordination transition. Complex coordination constraints are expressed by combining multiple semantic relationships using ports, and are configured using a comprehensive context-aware expression framework.

For future work, the operational semantics of coordination processes are the main focus, as coordinating multiple concurrently running instances poses unique challenges. In particular, the multi-instance nature of the run-time requires that semantic relationships are instantiated multiple times, once for each context. This leads to a highly complex instance representation of a coordination process at run-time, which must be kept synchronized with each process instance in the relational process structure and the execution status of each process instance. Both operational semantics and large process structure coordination will be investigated in future work. Additionally, a thorough empirical investigation of the coordination process modeling concept shall demonstrate its applicability in practice.

Acknowledgments. This work is part of the ZAFH Intralogistik, funded by the European Regional Development Fund and the Ministry of Science, Research and the Arts of Baden-Württemberg, Germany (F.No. 32-7545.24-17/3/1)

References

1. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of GSM-Based Artifact-Centric Systems through Finite Abstraction. In: 10th Int'l Conf. on Service-Oriented Computing (ICSOC). pp. 17–31. Springer (2012)

2. Damaggio, E., Hull, R., Vaculín, R.: On the Equivalence of Incremental and Fixpoint Semantics for Business Artifacts with Guard–Stage–Milestone Lifecycles. Information Systems 38(4), 561–584 (2013)
3. Decker, G., Weske, M.: Interaction-centric Modeling of Process Choreographies. Information Systems 36(2), 292–312 (2011)
4. Deutsch, A., Li, Y., Vianu, V.: Verification of Hierarchical Artifact Systems. ArXiv e-prints (2016)
5. Eid-Sabbagh, R.H., Dijkman, R., Weske, M.: Business Process Architecture: Use and Correctness. In: 10th Int'l Conf. on Business Process Management (BPM). pp. 65–81. Springer (2012)
6. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Many-to-Many: Some Observations on Interactions in Artifact Choreographies. In: 3rd Central-European Workshop on Services and their Composition (ZEUS), 2011. CEUR Workshop Proceedings, vol. 705, pp. 9–15. CEUR-WS.org (2011)
7. Hull, R., Damaggio, E., de Masellis, R., Fournier, F., Gupta, M., Heath,III, Fenno Terry, Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P.N., Vaculín, R.: Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In: 5th ACM Int'l Conf. on Distributed Event-based System (DEBS), 2011. pp. 51–62. ACM (2011)
8. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath,III, Fenno Terry, Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P.N., Vaculín, R.: Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles. In: 7th Int'l Workshop on Web Services and Formal Methods (WS-FM) 2010. LNCS, vol. 6551, pp. 1–24. Springer (2011)
9. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a Framework for Object-aware Process Management. Journal of Software Maintenance and Evolution: Research and Practice 23(4), 205–244 (2011)
10. Müller, D., Reichert, M., Herbst, J.: Data-driven Modeling and Coordination of Large Process Structures. In: 15th Int'l Conf. on Cooperative Information Systems (CoopIS). pp. 131–149. LNCS, Springer (2007)
11. Müller, D., Reichert, M., Herbst, J.: A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures. In: 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE). pp. 48–63. LNCS, Springer (2008)
12. Nigam, A., Caswell, N.S.: Business Artifacts: An Approach to Operational Specification. IBM Systems Journal 42(3), 428–445 (2003)
13. Object Management Group: Business Process Model and Notation (BPMN), Version 2.0 (2011)
14. Steinau, S., Andrews, K., Reichert, M.: The Relational Process Structure. In: 30th Int'l Conf. on Advanced Information Systems Engineering (CAiSE). pp. 53–67. Springer (2018)
15. Steinau, S., Künzle, V., Andrews, K., Reichert, M.: Coordinating Business Processes Using Semantic Relationships. In: 19th IEEE Conf. on Business Informatics (CBI). pp. 33–43. IEEE Computer Society Press (2017)
16. Sun, Y., Xu, W., Su, J.: Declarative Choreographies for Artifacts. In: 10th Int'l Conf. on Service-Oriented Computing (ICSOC). pp. 420–434. Springer (2012)
17. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A Framework for Lightweight Interacting Workflow Processes. International Journal of Cooperative Information Systems 10(04), 443–481 (2001)
18. van der Aalst, W.M.P., ter Hofstede, Arthur H. M., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)