Coordinating Business Processes Using Semantic Relationships

Sebastian Steinau, Vera Künzle, Kevin Andrews, and Manfred Reichert
Institute of Databases and Information Systems
Ulm University, Germany
{sebastian.steinau,vera.kuenzle,kevin.andrews,manfred.reichert}@uni-ulm.de

***Abstract*—In enterprises, different business processes collaborate to achieve a common business goal. The processes involved in such a collaboration are connected to each other in one-to-one, one-to-many, and many-to-many relationships. The complex interdependencies between these processes require proper process coordination. Current approaches addressing process coordination rely on message exchanges between the interacting processes with focus on syntax, while neglecting the semantics of these message exchanges between multiple processes. This paper introduces *semantic relationships*, a concept that provides the means to model process coordination based on message semantics, resulting in a high level of abstraction and thus facilitating process coordination. Semantic relationships incorporate the support for one-to-many and many-to-many process relations and affect process execution solely if required, allowing for concurrent and asynchronous process execution.**

***Index Terms*—Business Process, Business Process Coordination, Semantic Relationships, Object-aware Process**

1. Introduction

Process-aware information systems (PAIS) support the modeling, execution and monitoring of individual business processes. In a general business setting, many different (sub)processes collaborate to achieve a business goal. Especially when creating a complex product or delivering a service, many different processes may be involved in achieving the business goal. In general, the processes may have numerous complex interdependencies, e.g., requiring that a process may only start after every dependent process has been completed. Additionally, at runtime, multiple instances of a process, each progressing differently, need to be considered and handled properly and consistently. For this, modeling a correct coordination of processes and enforcing it at runtime is crucial for achieving the business goal.

Ideally, in a PAIS, the supported processes run concurrently and asynchronously where possible. Occasionally, however, the progress of an individual process may depend either explicitly or implicitly upon the progress of other processes. The required communication between the processes is denoted as a *process interaction* [1]. In order to comply with these dependencies, interacting processes must be coordinated. Thereby, any coordination effort should affect the concurrent execution of the processes as little as possible. Many current coordination approaches, however, only consider the coordination between two individual processes, neglecting one-to-many or many-to-many relationships. These relationships prove especially challenging at runtime, as the number of processes and the interdependencies may change or evolve dynamically (cf. Fig. 1).

In the predominant activity-centric process support paradigm, message exchanges between processes are used for coordination purposes (e.g., [2], [3]). As this paradigm is imperative in nature, messages are modeled with a focus on syntax. Message exchanges (i.e., process interactions), therefore, do not consider primarily the semantics of the exchanged messages. A thorough analysis of a significant number of process models as well as informal process descriptions revealed that message semantics can be classified based on the intended purpose of the messages. More precisely, five distinguishable patterns were observed that define the fundamental semantics of a message and, hence, define the respective process interactions as well. Based on these findings and taking additional requirements for process coordination into account, a basic concept for process coordination, denoted as *semantic relationships,* was developed.

Semantic relationships constitute a high-level concept that allows specifying coordination constraints for concurrently running processes based on five patterns. Semantic relationships can be configured in various ways to represent complex dependencies between processes. The benefits of this approach comprise reduced modeling efforts and improved process model maintenance by separating coordination modeling from the actual process models. Note that this allows for asynchronous process execution as well. Semantic relationships originated in the object-aware process management approach [4], [1], [5]. This paper expands upon previous work by introducing additional semantic relationships and clarifying the benefits of this coordination concept. It generalizes semantic relationships to achieve independence from the object-aware process model. In principle, semantic relationships can be used to coordinate processes modeled in any paradigm.

The remainder of the paper is organized as follows. Section 2 discusses challenges and requirements related to process coordination. Section 3 introduces the concept of semantic relationships. The particular challenge of asynchronously executing processes is explained in Section 4.

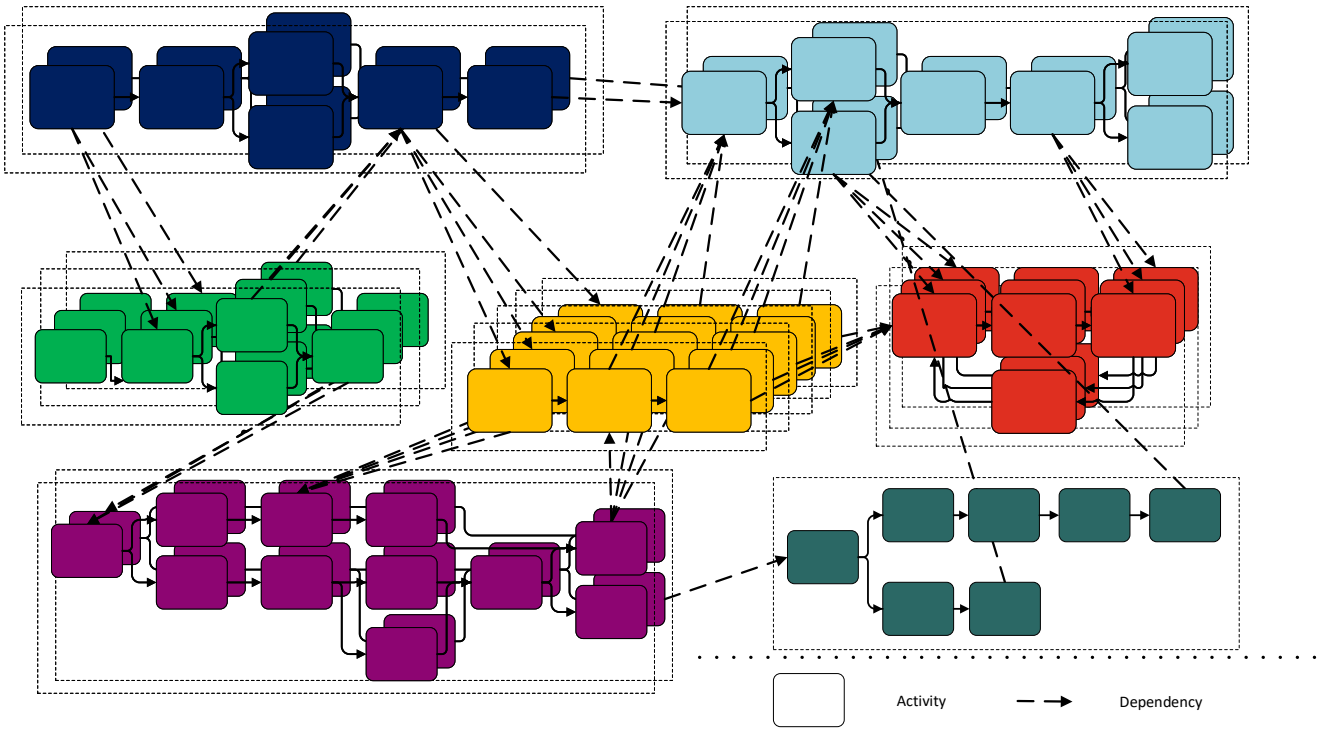

Figure 1. Processes and their dependencies at runtime

Section 5 presents evaluation results. Section 6 discusses related work before concluding the paper with a summary and an outlook in Section 7.

2. Problem Statement and Requirements

For illustration purposes, a process from the medical domain is used as a running example. The process is a simplified *diagnosis and treatment process* at a hospital.

Example 1 (*Simplified diagnosis and treatment process*)**.** *First, the patient needs to be admitted to the hospital. In this context, he must provide his name, address, symptoms, and a medical history listing allergies as well as previous diagnoses and treatments. Afterwards, the patient is sent to a physician for examination. She examines the patient and consults his medical history to determine potential causes of the observed symptoms. The physician may order tests at the hospital lab that checks for the presence of suspected diseases. She may further perform any other kinds of tests. Each test is conducted in a separate process. If a cause has been identified, a final diagnosis can be made and the patient advances to treatment. The physician starts the specific treatment process that administers the treatment according to the identified cause, e.g., giving antibiotics in case of a bacterial infection. After his recovery, the patient is discharged and leaves the hospital.*

While it might not be immediately apparent from the simplified running example, process coordination constitutes a complex and delicate endeavor with many intricate aspects that require careful considerations. In general terms, process coordination requires the specification and enforcement of *coordination constraints*.

Definition 1 (Coordination Constraint)**.**

A coordination constraint is a formal or informal statement describing one or more conditions or dependencies that exists between processes.

As an example of a coordination constraint, a *treatment process* may only be started if the patient has tested positive for the disease. As another example, a payout from an insurance claim may only be granted if the insured has a valid policy covering the damages and the damage assessment by an expert approves the claim. In this case, a treatment, test, expert assessment, or payment to the insured are considered to be separate processes. In order to implement such coordination constraints the processes need to interact.

Messages are widely used for information exchange in any kind of computer system. PAISs are no exception and, consequently, they rely on messages to coordinate processes and to exchange information between them; e.g., BPMN uses message flows for this purpose as well. However, BPMN presumes an imperative modeling style, and messages are mostly modeled regarding syntax. The purpose behind a message is left ambiguous and must often be inferred from context. While this might not be a problem for a process modeler, subject matter experts, who have to

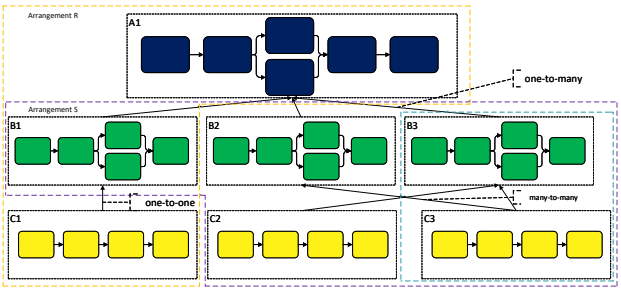

Figure 2. Abstract examples of arrangements

understand and correctly interpret the process model, might require significant effort. Therefore, it is highly desired that methods help to understand the semantics of the process coordination. Therefore, the following requirement is defined:

Requirement 1 (*Semantic Indications*)**.**
The coordination concept must indicate the semantics of the expressed coordination constraints.

In reasonably large business settings, processes may be dependent on another process. Processes having a connection are called *related* (have a *relation*), e.g., "test" and "treatment" processes are related. In this context, it is crucial to distinguish between *process type* and *process instance*. For example, process type A may have A_1,A_2..A_n as corresponding process instances. Subsequently, the term process is used when something applies to both type and instances. Moreover, two or more related process instances are said to be in an *arrangement* (cf. Fig. 2), e.g., one order may require several shipments.

Definition 2 (Arrangement)**.**
An arrangement is a collection of process instances that have possibly different types and are connected by relations between the process instances. The number of process instances of each type and their relations in an arrangement are fixed.

Changing an arrangement by, for example, adding or deleting a process or creating a new relation results in a new arrangement, i.e., an arrangement may be evolving. Moreover, the same process instance may be involved in different arrangements, e.g., process instance B_1 in Figure 2 is part of arrangements R and S. In general, coordination constraints may describe processes in an arrangement comprising one-to-one, one-to-many, or many-to-many relationships. This requires careful management of *process relations* to know which processes are connected and which are dependent on which other processes, i.e., knowledge of their arrangement is required.

However, the exact number of related process instances participating in an arrangement might not be known at design time, but only becomes known at runtime. To further complicate matters, at runtime, the number of process instances can often not be fixed either, as instances may be created or deleted dynamically. Additionally, the relations of dependent process instances may change dynamically. For example, an applicant for a job offer related to the position of "Software Architect" may fit better to the position of "Test Engineer". Accordingly, the application is handed over, i.e., the application process is then related to the job offer process "Test Engineer" instead of the job offer process "Software Architect". In general, such an overall *process structure* is continuously evolving (cf. [6], Fig. 1).

Definition 3 (Process Structure)**.**
A process structure constitutes an arrangement that comprises all interrelated process instances.

The evolving process structure affects the coordination constraints of both job offers, i.e., constraints might no longer be satisfied or now become satisfied. Furthermore, it is possible that processes do not have a direct relation, but still have transitive interdependencies. These must be taken into account and the processes need to be coordinated properly (cf. Requirement 2).

Requirement 2 (Relation Cardinality)**.**
The coordination concept must handle, possibly transitive, one-to-one, one-to-many, and many-to-many process relations, allowing for dynamic changes to the arrangements at runtime as well.

In imperative process models, each eventuality of the process must be explicitly modeled. An example of an eventuality is a decision that changes the outcome of a process. In general, related processes depend on such eventualities and, therefore, require a notification message to coordinate accordingly. Additionally, for executable process models, it must be ensured that sender and recipient both understand each other, i.e., the message format must be exactly specified and logic on each side to handle the message must be provided. For large processes with many eventualities, this might cause an enormous modeling effort. To reduce this effort, coordination modeling should abstract from the modeling of individual messages. Instead it should provide *generic, high-level modeling constructs for specifying coordination constraints*. The message exchanges required to actually implement these constructs should then be derived automatically and implicitly. In consequence, the message exchanges thereby may be hidden and are of no concern to the modeler. This *generic specification* is presented in Requirement 3:

Requirement 3 (Generic Specification)**.**
The coordination concept must provide generic high-level constructs for specifying coordination constraints. Necessary message exchanges should be implicit.

In a modern PAIS, processes may run concurrently to each other. With regard to process coordination, this concurrency should be affected as little as possible: i.e., waiting times due to unfulfilled dependencies should be kept minimal. In other words, process coordination should only occur at certain points in time during process execution, particularly if being indispensable for reaching the business goal. In general, an asynchronous process execution is

desired. For example, a physician may examine a patient and take a blood sample. This sample is then given to the laboratory for analysis. In a synchronous coordination, the physician must wait for the lab results before she may continue with the examination. In an asynchronous setting, instead of waiting for the lab results, the physician continues with examining the patient. The lab results will then simply arrive at a future point in time. If the lab analysis is finished after the patient examination, the physician must wait for the lab results after completing the examination before she may proceed with treating the patient. Asynchronous process execution is significantly complicated due to the fact that processes may be dynamically added or removed at runtime. Newly emerging processes need to become aware of the status of their related processes to properly evaluate their coordination constraints. The same applies to the existing processes, which need to reevaluate their coordination constraints when a new process emerges.

Requirement 4 (Asynchronous Concurrency)**.**
The coordination concept must allow for asynchronous and concurrent process execution. Coordination should affect process execution only when necessary.

In summary, any concept for coordinating processes should provide a generic, abstract way of specifying coordination constraints that allows for asynchronous execution and handles the necessary message exchanges automatically. Section 3 presents the concept of semantic relationships, which we consider as the foundation for a coordination approach fulfilling the four stated requirements.

3. The Concept of Semantic Relationships

In order to derive a concept for process coordination satisfying Requirements 1–4, existing graphical process models were analyzed with respect to the semantics of the process interactions. The BPMN process models with five message exchanges on average were created by the IT department of Ulm University. Hence, process interactions were modeled in terms of message exchanges. Additionally, the analysis included processes described informally in prose to include coordination scenarios that cannot be expressed graphically yet. The analysis revealed that with regard to the semantics of the process interactions as well as the associated coordination constraints, the interactions can be classified into five different categories. Accordingly, the semantic relationships reflecting these categories are denoted as *top-down, bottom-up, transverse, self,* and *self-transverse.* In particular, the five semantic relationships refer to the support of one-to-many relations between processes. Many-to-many relations may be expressed as several one-to-many relations, which then can be coordinated using semantic relationships (cf. Req-2).

Coordination constraints are represented using semantic relationships. Several, possibly different, semantic relationships might be required to adequately represent a coordination constraint. Consequently, a semantic relationship is required to have a logical value, i.e., true or false, at runtime. The logical value, in turn, indicates whether a semantic

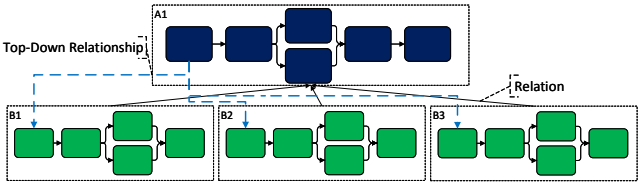

Figure 3. Schematic example of a top-down semantic relationship

relationship is satisfied or unsatisfied at a specific point in time during process execution . The semantic relationships involved in the representation of a coordination constraint may be combined with boolean operators to calculate the overall logical value of the coordination constraint. Based on this logical value, certain actions (e.g., continuing with process execution) are either allowed or prohibited.

Before specifying the various semantic relationships, two definitions must be provided. The terms *lower-level* and *higher-level* refer to the fact that the one-to-many relations are directed (i.e., represented by a directed edge, opposed to, e.g., relations in entity-relationship-diagrams). Many-to-many relationships may be broken down to several one-to-many relations. In turn, the direction of the relations induces a hierarchy between processes. A 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. Though strictly speaking not being necessary for semantic relationships themselves, the hierarchy between processes provides advantages for the automated determination of semantic relationships between processes and the technical implementation of semantic relationships. In the following, the five types of semantic relationships are presented in detail.

3.1. Top-down Semantic Relationship

A *top-down semantic relationship* exists if the execution of (possibly several) lower-level processes depends upon the execution status of a common higher-level process. Schematically, this is depicted in Figure 3, where process instances B_1--B_3 depend on A_1, i.e., process instance A_1 must reach a particular processing state before process instances B_1--B_3 may start/resume their execution. For example, the patient examination takes place after admitting the patient. Only after admission, a physician may order specific tests (test processes are instantiated) at the lab or perform any other examination required. If the patient has not been admitted yet, the physician must not perform any examinations (which is also not possible for practical reasons).

A top-down semantic relationship provides only the essential semantics to represent coordination constraints. To be able to properly represent coordination constraints, semantic relationships can be customized through configuration. For top-down relationships, the predominant question is when

the semantic relationship is no longer satisfied. In the example, after examining the patient and determining a final diagnosis, the physician proceeds with treating the cause of the illness. Further examinations are then no longer useful. Consequently, it must not be allowed to order additional tests. A top-down relationship, therefore, includes means to specify at which point the semantic relationship is satisfied for the first time and when it will no longer be satisfied.

3.2. Bottom-up Semantic Relationship

As opposed to top-down semantic relationships, a *bottom-up semantic relationship* is defined by one higher-level process being dependent on the execution status of one or more lower-level processes. Regarding the running example, lower-level processes include the *test processes* the physician ordered during the examination of the patient. Before a final diagnosis can be made, each lower-level process must complete and deliver a result (see Figure 4). Each of the process instances B_1-B_3 must reach a certain point in its enactment before higher-level process instance A_1 may resume.

Regarding configuration options, bottom-up relationships are more complicated than top-down relationships. To configure a bottom-up relationship, information on the set of lower-level processes is required. In the running example, the different tests ordered by the physician produce a result, i.e., the patient either has a specific disease or not. A coordination constraint of the running example states that all tests have to complete before the higher-level process may proceed to the final diagnosis. Obviously, it might make more sense to proceed if a test has produced a positive result, i.e., the patient suffers from the disease.

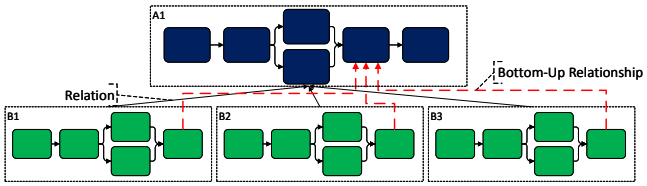

Figure 4. Schematic example of a bottom-up semantic relationship

In the running example, it is assumed that the patient has no other diseases, therefore all other tests are irrelevant and the physician may make the diagnosis. In general terms, a bottom-up relationship is satisfied once a subset of lower-level processes meets a condition. The size of the subset and the condition are specified by a coordination constraint.

To specify such a condition, a bottom-up relationship needs an expression framework. The framework must be able to access process data, e.g., to determine which activities have been executed or which values certain data elements posess. This information can then be combined with comparison operators, boolean operators, and other functions to create an expression representing the desired condition. The expression framework determines the extent to which complicated coordination constraints can be represented. Depending on the coordination constraint, expressions might become overly complex, to the point where they are no longer easy to understand. The concepts presented in Section 5 can be used to alleviate the problem by providing abstractions that simplify the specification of the conditions.

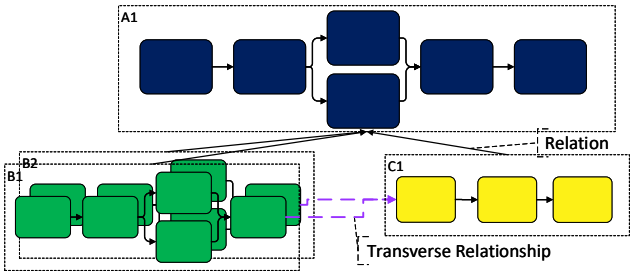

Figure 5. Schematic example of a transverse semantic relationship

3.3. Transverse Semantic Relationship

Transverse semantic relationships coordinate two sets of processes of different type in the context of a common higher-level process. In particular, the execution progress of processes in one set depends on the execution status of processes from the other set (cf. Fig. 5). The process instance C_1 depends on process instances from set $\{B_1, B_2\}$. However, C_1 has no direct or transitive relation with B_1 or B_2. Instead, B_1 and B_2 are related indirectly to C_1 as they are lower-level processes of process instance A_1. The common higher-level process of the processes from the two sets of the transverse semantic relationship is denoted as *common ancestor*, which is A_1 in this case. Note that each transverse semantic relationship has exactly one common ancestor. A common ancestor and the other involved processes may be related transitively.

In the running example, *test* and *treatment processes* have a transverse relationship. A *treatment process* may only start if the corresponding *test process* is completed and the result is positive, i.e., the patient suffers from the specific disease. A *treatment process* may also depend on several *test processes*, i.e., if the tests have a chance of producing false positives or false negatives. As a consequence, the same type of test must be performed multiple times until a reasonable level of confidence in respect to the result is achieved.

A transverse semantic relationship requires an expression framework in order to specify a condition. The framework and the condition serve the same purpose as in the context of bottom-up semantic relationships. Further, the framework does not require additional functionality specific to transverse semantic relationships. In the running example, a coordination constraint states that at least one *test process* must have a positive result before a corresponding *treatment process* may be started. This corresponds to a transverse semantic relationship with an expression representing the "at least one test with a positive result" condition.

Additionally, transverse semantic relationships allow for the specification of a common ancestor. The common an-

cestor significantly influences the transverse relationship as well as the representation of the coordination constraint. In the running example, the *diagnosis and treatment process* is the common ancestor in the transverse relationship between *test process* and *treatment process*. As example, assume that a *hospital process* is added as a higher-level process of the *diagnosis and treatment process*, i.e., the *hospital process* has a one-to-many relation with the *diagnosis and treatment process*.

If the modeler chooses to change the common ancestor from the *diagnosis and treatment process* to the *hospital process*, it alters the meaning of the transverse relationship, provided the condition of having one positive test for a disease to start treatment remains the same. The sets of the transverse relationship now contain all *test* and *treatment processes* of the entire hospital. Due to the context change, in turn, *test* and *treatment processes* cannot be easily associated with a specific patient. As a consequence, for example, it would be allowed to treat a patient for the flu if any other patient was tested positive for it. The choice of context for a semantic relationship, i.e., the common ancestor, should be carefully considered to avoid undesired effects in process coordination.

3.4. Self / Self-Transverse Semantic Relationship

A *self semantic relationship* constitutes the simplest semantic relationship, which describes a dependency between two parts of the same process, e.g., patient examination requires the completion of patient admission. Normally, this semantic relationship is captured implicitly in the process models, e.g., in activity-centric models, the *self* semantic relationship is expressed through control flow. As a result, the self semantic relationship possesses no configuration options. However, a process might also depend on other processes of the same type, as opposed to depending on parts of itself (*self* relationship) or processes of different type (top-down, bottom-up, or transverse relationships). As example assume that there are *test processes* being mutually exclusive. This can be the case if the testing requires the patient to be injected with a substance. Other substances used in other tests might be incompatible due to various reasons, e.g., blood clotting causing thrombosis. Therefore, if one test has been performed, other tests must not be performed anymore.

Self-transverse semantic relationships, in turn, are provided to express such coordination constraints. A self-transverse semantic relationship expresses a constraint between processes of the same type, whereas a transverse semantic relationships expresses a constraint between processes of different types. In general, a self-transverse relationship corresponds to a choice or m-out-of-n pattern. The sets of processes are defined analogously to transverse semantic relationships. Additionally, they provide the same configuration options as transverse relationships, i.e., an expression framework for specifying conditions and the choice of a common ancestor. Schematically, a self-transverse semantic relationship is depicted in Figure 6, where process

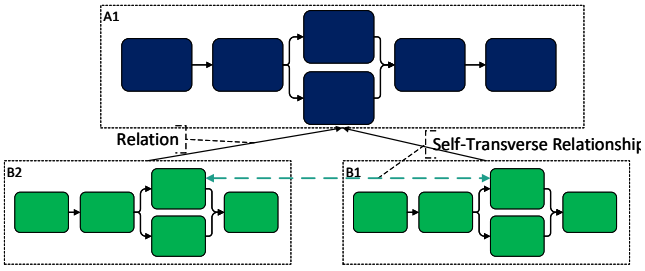

Figure 6. Schematic example of a self-transverse semantic relationship

instance A_1 serves a common ancestor to process instances B_1 and B_2 in a self-transverse relationship.

All semantic relationships are closely linked to relations between processes. On one hand, creating a semantic relationship establishes a relation between processes as well. On the other, it is feasible to automatically derive a semantic relationship from the relation between two types of processes. Note that this provides a significant advantage when modeling process coordination with semantic relationships, as it reduces modeling efforts in case the relations between processes are known. Another benefit of semantic relationships for a process modeler is the focus on the business goal itself and less on the way how to achieve the goal, which fosters a more declarative modeling style. Regarding the requirements elaborated in Section 2, the presented semantic relationships fulfill Requirements 1-3. In Section 4, state-based abstractions are presented, which additionally allow fulfilling Requirement 4.

4. State-based View and Asynchronous Process Execution

According to Requirement 4, coordination should only take place at specific points in time. Between its points of coordination, in turn, a process should run asynchronously to other processes. Though, in principle, an asynchronous process execution is already possible with semantic relationships, the currently used notation is unable to display those parts of a process that can be asynchronously executed in respect to other processes. Figures 3-6 depicts a single activity as the point of coordination, whereas the coordination is actually possible in regard to several activities. For example, during patient examination, the physician may perform several activities herself. However, instead of ordering a test at the lab (i.e., start a new test process) only after one specific activity, she may order a test after each of these activities. Furthermore, the notation incorporates semantic relationships into the existing process models, which may prove disadvantageous, i.e., require increased modeling efforts when changing the models later. In order to tackle this challenge, we developed *state-based views.*

State-based views provide a simplified view on a process by abstracting from (i.e., hiding away) unnecessary details. Visible from the outside are only the *states* of the process,

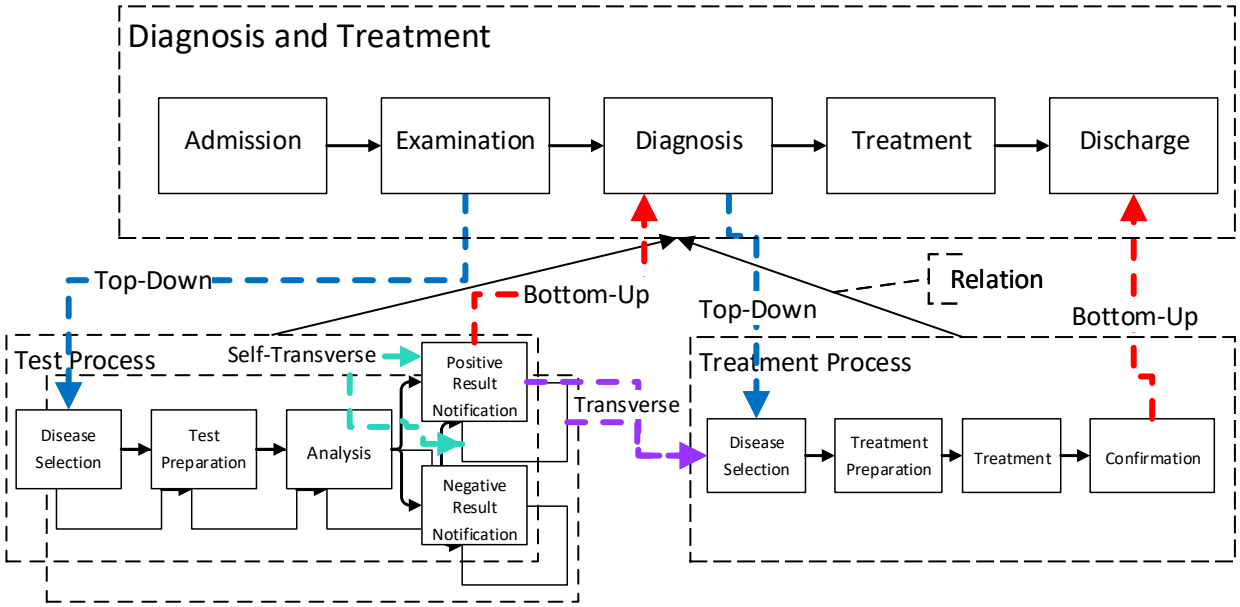

Figure 7. State-based views of the processes in the example

which, therefore, must convey all the necessary information. Regarding the running example, parts "Admission", "Examination", "Diagnosis", "Treatment" and "Discharge" (cf. Example 1) constitute possible states. States represent logical groupings of process elements, like, for example, activities in activity-centric processes. Additionally, states represent results of a process, i.e., when making decisions, the process enters one of several mutually exclusive states. For example, the *test processes* in the running example either have a "positive" or a "negative result". Figure 7 reflects the state-based views of the processes from Example 1.

The partitioning of process models into states is, in principle, arbitrary. Hence, there may be more than one possibility for a reasonable partitioning. However, states must be chosen with regard to the coordination of the processes, as states are the only visible information for other processes. Furthermore, the intrinsic logical grouping of underlying process elements, (e.g., activities) should not be neglected. For semantic relationships, states provide an easy-to-access interface for the processes. More precisely, top-down, bottom-up, transverse, self, and self-transverse semantic relationships can define process dependencies as well as coordination constraints based on the states of the processes. At runtime, states are either active, non-active or have been active at one point in time before. Regarding the running example, a coordination constraint for the running example states that a physician may order tests only during patient examination. In terms of states and semantic relationships, this represents a top-down semantic relationship between the *diagnosis and treatment process* and the *test process*. If state "examination" of the *diagnosis and treatment process* becomes active, the semantic relationship allows *test processes* to be started, i.e., state "disease selection" of the *test process* may become active as well.

Concerning Requirement 4, states provide the means to coordinate processes at specific points in time and only if necessary. A blocked execution caused by unsatisfied coordination constraints may only occur when transitioning from one state to another. Within a state, in turn, a process is executed asynchronously to other processes without interference from the coordination. Furthermore, depending on the presence and status of coordination constraints for specific states at runtime, asynchronous execution is possible over multiple states as well. For example, states "Test Preparation", "Analysis", "Positive Result Notification", and "Negative Result Notification" of the *test process* are not subject to coordination constraints. The activities within these states can be executed asynchronously to the activities within the "Patient Examination" state. (cf. Figure 7).

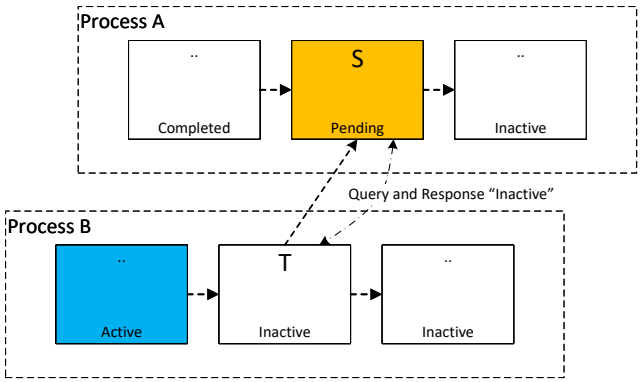

Figure 8. Asynchronous Execution with Dependency: Process A cannot activate state S, state T is inactive

The asynchronous execution poses challenges to the correct coordination of dependent processes, which are illustrated in the following.

Example 2. Suppose there are two processes A and B running concurrently and asynchronously to each other. Process A has state S_A, which has a semantic relationship with process B with state T_B; i.e., A must wait for T_B to become active before its state S_A can become active as well. As both processes are executed asynchronously and concurrently, either process A or B may reach its respective state faster. In case process A is faster (cf. Fig. 8), upon reaching state S_A, a query to T_B is issued for its status; T_B returns status "Inactive". Consequently, state S_A must wait for state T_B to become active. As soon as T_B becomes active, state S_A is notified and activated. If T_B is already active (cf. Fig. 9), the query returns "Active" for the status of T_B and state S_A may immediately activate. In case process B is faster (cf. Fig. 9), state T_B may be already active when A reaches state S_A. If T_B has notified process A of its status and A stored the notification, subsequently S_A can activate immediately.

A semantic relationship can solve this challenge by providing a mutual publish-subscribe between both processes. The actual message exchanges are completely hidden. State-based views offer additional benefits for semantic relationships. The abstraction provided by them fosters the process modeling of semantic relationships, as the process modeler must not concern himself with details of the coordinated processes. State-based views provide a clear and simple foundation for specifying semantic relationships, and, thus, for reducing the complexity of the modeling.

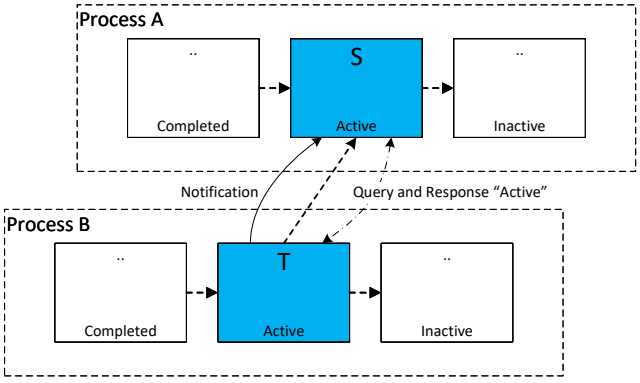

Figure 9. Asynchronous Execution with Dependency: Process A can activate state S, state T is active

With the state-based abstraction, it becomes possible to create coordination models with semantic relationships that exist separately from the coordinated processes. As advantage, existing models need not be changed, as the state-based view can be realized "on top" of existing process models. This allows modeling semantic relationships in two phases:

1) The normal modeling phase, without any concerns for coordination.
2) The coordination is brought into play and modeled around the existing process models in a separate model.

It is expected that this clear separations of concerns makes initial model creation easier and also fosters process model maintenance.

5. Evaluation

Semantic relationships were derived by analyzing a set of process models. As a first evaluation, semantic relationships were applied to a subset of the same process models from the initial analysis to check whether semantic relationships can be applied. The subset consisted of administration processes from Ulm University modeled in terms of BPMN. The processes were chosen due to their high quality and their high number of process interactions, i.e., message exchanges between BPMN pools. In total, the 18 process models comprised 92 message exchanges. We were able to fully replace the message exchanges with appropriate semantic relationships, providing evidence that they can be applied to existing process models. However, as in the context of BPMN process interactions correspond to one-to-one relations between processes, meaningfulness is limited. The evaluation cannot show the benefits for one-to-many or many-to-many process relationships. For several processes, a redesign of specific parts would foster the interaction modeling based on semantic relationships.

In addition to the application to the BPMN models, we specified several real-world processes from the insurance, human resource, and university domains. In particular, the modeled processes included a damage claim process from an insurance company, a job application process, and a process from an HR software managing absence from work for employees. All process models included coordination constraints with one-to-many relations between processes, e.g., the damage claim may require multiple expert assessments of the damages.

The processes were modeled using the object-aware process modeling approach [4], [1], which provides process coordination support based on semantic relationships and state-based abstraction. The PHILharmonicFlows prototype [7] comprises a modeling tool with which the processes were modeled. For all process models, the coordination constraints specified by the process descriptions could be faithfully reproduced in the model using the presented semantic relationships, with two exceptions. In this particular cases, the expression framework did not provide the necessary functionality for specifying the appropriate condition for a bottom-up relationship. Although the coordination constraint could not be reproduced, it is not a conceptional problem of semantic relationships, but one of the expressiveness of the used framework.

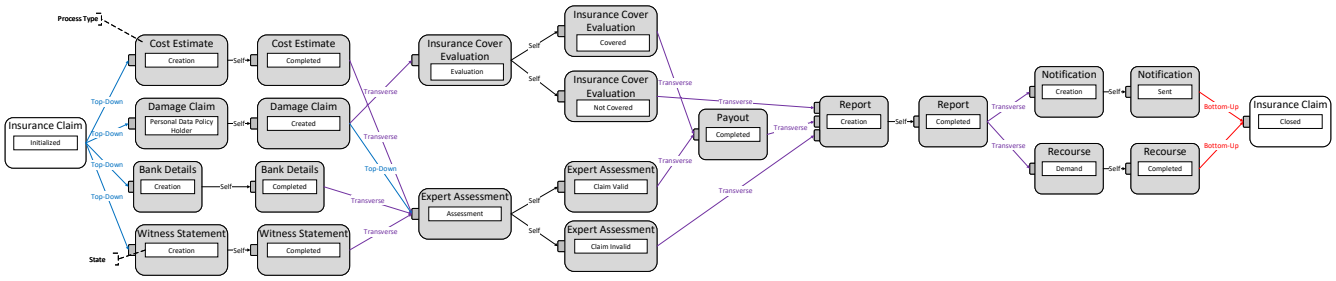

Figure 10. Insurance Claim Coordination Process

For example, the insurance claim model[1] uses 14 different process types with multiple one-to-many relations. Figure 10 shows a *coordination process* (also known as a macro process, cf. [1]). The coordination process shows process types and their states connected by semantic relationships. The coordination required 18 semantic relationships in total (self relationships are not counted due to triviality). Since modeling object-aware processes produces executable process models that can be executed using the PHILharmonicFlows Runtime Tool [8], test cases were designed to which the process models were subjected. Special focus was put on the proper coordination of processes in one-to-many arragements. The test cases could be successfully executed, satisfying the coordination constraints set by the process specification despite efforts to break them. Overall, the conducted evaluations proved the feasibility of using semantic relationships in process coordination. However, the understandability and practicability in modeling must be thoroughly evaluated in future user studies.

6. Related Work

Regarding the activity-centric paradigm, several approaches enable a specific kind of coordination. In [9], [10], multiple instance workflow patterns for coordinating processes are described. The business process architecture approach [11], [12] identifies generic patterns to describe a coordination between processes. Moreover, BPEL4Chor [13] extends BPEL by adding coordination support in the form of process choreographies. iBPM [2], [14] enhances BPMN to support coordination of processes by modeling process interactions.

Common to all these approaches is the use of messages as a mechanism for coordination. While the exchange of messages allows for a fine-grained process coordination, all message flows have to be identified, the contents of the messages be defined, and the message recipients be determined. This introduces an enormous complexity when facing multiple processes that need to be coordinated. In many cases, it impairs the flexible execution of the processes involved. Except Proclets, the modeling of coordination aspects is integrated with the actual process models, adding complexity to the process models. Changing a process model then requires additional efforts, as changes may have a substantial effect on other parts of the model, which have to be adapted as well. Strictly separating coordination mechanism on one hand and the processes on the other, therefore, could help to reduce the effort required for evolving the model. Additionally, none of the approaches fully supports Requirements 1-4, i.e., *Semantic Indications, Relation Cardinality*, *Generic Specification,* and *Asynchronous Concurrency*.

Proclets [3], [15] are lightweight processes with focus on process interactions as well. They interact via messages called *Performatives*. Proclets allow specifying the cardinality for a message multicast, i.e., the number of Proclets that receive a performative. However, this number is fixed at design time, so Requirement 2 is only partially fulfilled. In contrast, Proclets are capable of asynchronous and concurrent execution. The other requirements are not considered by the Proclet approach.

Case handling [16], [17], [18] and artifact-centric process management [19] use the Guard-Stage-Milestone (GSM) meta-model [20], [21] for process modeling. Central to these approaches is the *case/artifact*, which holds all process-relevant information. It may further interact with other cases or artifacts. However, GSM does not provide dedicated coordination mechanisms, but incorporates a sophisticated expression framework, that, in principle, allows creating the needed coordination mechanisms with expressions. As a drawback, these expressions might become very complex and explicitly need to be integrated into the process model. Therefore, model verification [22], [23], [24] constitutes an important aspect of artifact-centric process management. Further, [25] recognizes the need for supporting many-to-many relationships in artifact-centric choreographies. Also, the challenge of dynamically emerging and disappearing processes at runtime is acknowledged (cf. Req-2). However, Requirements *Semantic Indications, Generic Specification,* and *Asynchronous Concurrency* (cf. Req-1, Req-3, Req-4) are not addressed.

The coordination of large process structures with focus on the engineering domain is considered in [6], [26]. The COREPRO approach explicitly considers process relations with one-to-many cardinality and dynamic changes at runtime (cf. Req-2), 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

1. The full process model is available at https://goo.gl/GuA79T

transitions do not take the semantics of the respective process interaction into account. In particular, transverse relationships between processes are not supported.

7. Summary and Outlook

Semantic relationships enable process coordination based on semantic criteria. The semantic relationships top-down, bottom-up, transverse, self, and self-transverse cover standard interaction patterns providing a basis for coordinating processes. To also represent more demanding coordination constraints, the basic semantic relationships may be configured. In particular, they are capable of handling a large number of process instances in various arrangements. Additionally, semantic relationships are able to accommodate dynamic changes to the overall process structure at runtime.

State-based abstractions (i.e., views) allow for asynchronous process execution and also provide a clear interface for the processes, which eases the configuration of a semantic relationship. Furthermore, the abstraction of a process with a state-based view, in principle, allows for the coordination of processes modeled in any paradigm or process modeling language with semantic relationships.

Future work will investigate the benefits of process coordination with semantic relationships. Several experiments and user studies are planned to evaluate practicability and understandability. Finally, a technical implementation of process coordination using the concept of semantic relationships will be presented.

References

[1] V. Künzle and M. Reichert, "PHILharmonicFlows: Towards a Framework for Object-aware Process Management," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, no. 4, pp. 205–244, 2011.

[2] G. Decker and A. Barros, "Interaction Modeling Using BPMN," in *Business Process Management Workshops 2007*. Springer, 2008, pp. 208–219.

[3] W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer, "Workflow Modeling using Proclets," in *7th Int'l Conf. on Cooperative Information Systems (CoopIS)*. Springer, 2000, pp. 198–209.

[4] V. Künzle and M. Reichert, "Towards Object-aware Process Management Systems: Issues, Challenges, Benefits," in *10th Int'l Workshop on Business Process Modeling, Development, and Support (BPMDS)*, ser. Lecture Notes in Business Information Processing. Springer, 2009, pp. 197–210.

[5] V. Künzle, "Object-Aware Process Management," Ph.D. dissertation, University of Ulm, 2013.

[6] D. Müller, M. Reichert, and J. Herbst, "Data-driven Modeling and Coordination of Large Process Structures," in *15th Int'l Conf. on Cooperative Information Systems (CoopIS)*, ser. Lecture Notes in Computer Science. Springer, 2007, pp. 131–149.

[7] C. M. Chiao, V. Künzle, K. Andrews, and M. Reichert, "A Tool for Supporting Object-Aware Processes," in *IEEE 18th Int'l Conf. on Distributed Object Computing - Workshops and Demonstrations (EDOCW)*. IEEE Computer Society Press, 2014, pp. 410–413.

[8] K. Andrews, S. Steinau, and M. Reichert, "A Runtime Environment for Object-Aware Processes," in *BPM Demo Session (BPMD)*. CEUR-WS.org, 2015, pp. 6–10.

[9] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[10] W. M. P. van der Aalst, "Workflow Patterns," in *Encyclopedia of Database Systems*. Springer, 2009, pp. 3557–3558.

[11] R.-H. Eid-Sabbagh, R. Dijkman, and M. Weske, "Business Process Architecture: Use and Correctness," in *10th Int'l Conf. on Business Process Management (BPM)*. Springer, 2012, pp. 65–81.

[12] R.-H. Eid-Sabbagh and M. Weske, "Analyzing Business Process Architectures," in *25th Int'l. Conf. on Advanced Information Systems Engineering (CAiSE)*. Springer, 2013, pp. 208–223.

[13] G. Decker, O. Kopp, F. Leymann, and M. Weske, "BPEL4Chor: Extending BPEL for Modeling Choreographies," in *IEEE Int'l Conf. on Web Services (ICWS)*, 2007, pp. 296–303.

[14] G. Decker and M. Weske, "Interaction-centric Modeling of Process Choreographies," *Information Systems*, vol. 36, no. 2, pp. 292–312, 2011.

[15] W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer, "Proclets: A Framework for Lightweight Interacting Workflow Processes," *International Journal of Cooperative Information Systems*, vol. 10, no. 04, pp. 443–481, 2001.

[16] H. R. Motahari-Nezhad and K. D. Swenson, "Adaptive Case Management: Overview and Research Challenges," in *IEEE 15th Conf. on Business Informatics*, 2013, pp. 264–269.

[17] H. A. Reijers, J. H. M. Rigter, and W. M. P. van der Aalst, "The Case Handling Case," *International Journal of Cooperative Information Systems*, vol. 12, no. 03, pp. 365–391, 2003.

[18] W. M. P. van der Aalst, M. Weske, and D. Grünbauer, "Case Handling: A New Paradigm for Business Process Support," *Data & Knowledge Engineering*, vol. 53, no. 2, pp. 129–162, 2005.

[19] A. Nigam and N. S. Caswell, "Business Artifacts: An Approach to Operational Specification," *IBM Systems Journal*, vol. 42, no. 3, pp. 428–445, 2003.

[20] R. Hull, E. Damaggio, R. de Masellis, F. Fournier, M. Gupta, Heath,III, Fenno Terry, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. N. Sukaviriya, and R. Vaculín, "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)*. ACM, 2011, pp. 51–62.

[21] R. Hull, E. Damaggio, F. Fournier, M. Gupta, Heath,III, Fenno Terry, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. N. Sukaviriya, and R. Vaculín, "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*, ser. Lecture Notes in Computer Science, vol. 6551. Springer, 2011, pp. 1–24.

[22] F. Belardinelli, A. Lomuscio, and F. Patrizi, "Verification of GSM-Based Artifact-Centric Systems through Finite Abstraction," in *10th Int'l Conf. on Service-Oriented Computing (ICSOC)*. Springer, 2012, pp. 17–31.

[23] E. Damaggio, R. Hull, and R. Vaculín, "On the Equivalence of Incremental and Fixpoint Semantics for Business Artifacts with Guard–Stage–Milestone Lifecycles," *Information Systems*, vol. 38, no. 4, pp. 561–584, 2013.

[24] A. Deutsch, Y. Li, and V. Vianu, "Verification of Hierarchical Artifact Systems," *ArXiv e-prints, arXiv:1604.00967*, 2016.

[25] D. Fahland, M. de Leoni, B. F. van Dongen, and W. M. P. van der Aalst, "Many-to-Many: Some Observations on Interactions in Artifact Choreographies," in *3rd Central-European Workshop on Services and their Composition (ZEUS)*, vol. 705. CEUR-WS.org, 2011, pp. 9–15.

[26] D. Müller, M. Reichert, and J. Herbst, "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)*, ser. Lecture Notes in Computer Science. Springer, 2008, pp. 48–63.