ulm university universität
uulm

Ulm University | 89069 Ulm | Germany

Faculty of Engineering and Computer Science
Institute of Databases and Information Systems

Discovery and Evaluation of Coordination Patterns for Business Processes in many-to-many Relationships

Master's Thesis at Ulm University

Submitted By:
Marisol Schwarz Rosado
marisol.schwarz-rosado@uni-ulm.de

Reviewer:
Prof. Dr. Manfred Reichert
Dr. Rüdiger Pryss

Supervisor:
Sebastian Steinau

2019

Version March 24, 2019

Satz: PDF-LaTeX 2$_\varepsilon$

Abstract

Today, organisations use process-oriented systems to manage and automate the enactment of their business processes. The cornerstone artifact is the process model, which at design-time is used to describe the steps that need to be fulfilled in order to reach a business goal. At run-time, the process model is executed and process instances are created. The existing modelling approaches are based on three main paradigms: the more traditional activity-centric paradigm, the case handling paradigm and the more recent data-centric paradigm.

Process models can be classified into *monolithic* and *interacting process models*. Monolithic process models are predominantly created in the activity-centric and case-handling paradigm. In a monolithic process model, all the involved resources and activities are contained in one vast model. In monolithic process models, interactions occur between the different partners involved in a cross-organisational setting which exchange messages with one another. Interacting process models are prevalent in the data-centric paradigm. In interacting process models, interdependent processes interact with one another such that on a meta-level a composite business process is achieved. In both types of models, interactions between interrelated processes need to be properly coordinated such that a common business objective can be reached.

Handling the complexity generated by highly interconnected scenarios, involving hundreds of processes, is a challenge in business process management. Process management systems for such collaborations must be capable of handling both synchronous and asynchronous process interactions. In the context of process management systems, different pattern catalogues such as the Service Interaction Pattern or Correlation Pattern have been used for describing fundamental types of interactions that repeatedly arise during business process modelling. Yet, until now, none of the existing pattern catalogues has explicitly tackled the interactions of heterogeneous business processes in a many-to-many relationship setting. Furthermore, the existing pattern catalogues for the interaction-perspective are not paradigm independent, but mainly focus on the activity-centric paradigm.

For modelling multiple interacting processes with different dependency constraints, a collection of patterns that explicitly describes interactions among processes in different types of relationships, in a paradigm-independent manner, is required. This thesis proposes a catalogue of patterns, named the Process Coordination Patterns, describing process interactions in a one-to-many and many-to-many relationship setting. In the developed pattern catalogue, the discovered seven patterns are illustrated by abstracting from any specific paradigm. The PCPs may be used as guidance for evaluating the degree to which existing approaches capture more complex process interactions. In this thesis, the proposed pattern catalogue is put into practice by evaluating the degree to which two modelling approaches, based on different paradigms, can support the seven Process Coordination Patterns.

Contents

1 Introduction

1.1. Motivation

The increasingly competitive business environment is compelling organisations to pursue various innovative means to reduce costs and satisfy customer demands. Organisations have recognised business processes as a target for optimisation in order to stay competitive and succeed in the market. For this purpose, Business Process Management (BPM) is a common practice followed by a large number of organisations in all business areas [21, 27].

The field of BPM stems from the notion of business processes as being pervasive, in other words business processes are fundamental components of modern organisations. The idea of a business process can be traced back to the Industrial Revolution where large-scale automation changed the concept of production. Individuals who formerly conducted processes on their own were arranged into groups of specialised workers. Each individual worked on single repetitive activities that formed part of an overall business process. The Industrial Revolution marked a transition for workers from generalists to specialists while the process itself became the means for coordinating the individual activities that led to the desired production output [43]. BPM recognises that due to their close relationship to a company's product or services, managing business processes adequately enables an organisation to faster respond to market changes and quickly adapt to business environments [5, 43].[1]

Over time, different paradigms have emerged on how to manage business processes. The main ones are the activity-centric, the case-handling and the data-centric paradigm. Within each paradigm, several approaches exist and new ones are constantly being proposed. Approaches keep emerging because they aim to capture business scenarios in a form that reflects real-world processes more accurately. Based on these approaches, today organisations use Process Management Systems (PrMS) for optimising and automating their business processes. The core element of those process-oriented systems is the process model, which at design-time describes the steps that need to be fulfilled in order to reach a business goal. At run-time, the process model is executed and

[1] In the literature some authors use the terms *business processes* and *processes* as synonyms. Tough, in the context of this thesis, business processes are related to business organisations, as they define how to achieve the goals of the organisation and thus they are a subset of the set of processes.

process instances are created [14, 28, 39]. In approaches that are part of the activity-centric paradigm a process model is composed of activities assigned to resources which are executed in a pre-specified order [13, 14, 43, 60]. In the case-handling paradigm, a process model is also illustrated with activities carried out by resources, however the process model allows more flexibility concerning the order in which activities are executed [20, 38, 56, 58]. In the data-centric paradigm, a process model consists of several interdependent data objects, each of them with a determined lifecycle that may interact with those of other data objects [6, 16, 25, 36]. [2]

Process models can be classified into *monolithic* and *interacting process models*. Monolithic process models are predominantly created in the activity-centric and case-handling paradigm. In a monolithic process model, all the involved resources and activities are contained in one vast model. In monolithic process models, interactions occur between the different partners involved in a cross-organisational setting which exchange messages with one another. Interacting process models are prevalent in the data-centric paradigm. In interacting process models, interdependent processes interact with one another such that on a meta-level a composite business process is achieved. In both types of models, interactions between interrelated processes need to be properly coordinated such that a common business objective can be reached [8, 29, 43, 54].

Monolithic process models have proven to be applicable in modelling business scenarios where the process flow is determined by activities that need to be executed in a predefined order such as accounting, insurance handling or use of municipal services [29]. In such scenarios, only a few interactions between single process participants need to be handled. However, when modelling business scenarios with numerous interactions as found in logistics, production and human resources management, monolithic process models may not represent them adequately. Describing the overall business process in one model may not properly capture the nature of numerous interacting processes that have different relationships with one another [34, 36]. Modelling scenarios with several interacting processes requires changing the perspective from one vast unfragmented process to rather elementary processes that interact with one another at certain points in time [47, 54].

In the IT domain, pattern catalogues such as the *Service Interaction Pattern [4, 55]* and *Correlation Pattern* [3] have been used as the highest level of abstraction for identifying and describing a number of recurrent characteristics of process interactions to be modelled. However, both pattern catalogues are not paradigm-independent and mainly focus on monolithic process models. The interactions are typically based on the exchange of messages between single process participants that are part of the overall process model. Consequently, those pattern catalogues do not fully cover numerous interactions among heterogeneous process instances in one-to-many *(1:n)*, many-to-one *(m:1)*, and many-to-many *(m:n)* relationships.

[2] A detailed description of the mentioned paradigms is beyond the scope of this thesis. The reader may be referred to the cited sources for details.

Hence, it is essential to look at interacting processes at a comprehensive level, in order to properly model scenarios with numerous interacting process instances in different types of relationships. A pattern catalogue that explicitly captures interactions of processes in different types of relationships in a paradigm-independent manner is needed. Even though some papers in the literature mention the problematic nature of capturing interactions among process instances that have relationships which go beyond one-to-one *(1:1)*, an appropriate pattern catalogue has not been proposed to date [16, 28, 51, 54]. This research gap arises from the fact that scenarios with numerous interacting processes have been mostly ignored by process designers. The reason for this is that popular process modelling paradigms do not focus on interactions.

Motivated by the existing research gap, this thesis aims to provide a collection of the different types of interactions that may exist among process instances that have relationships with one another of the type *1:n*, *m:1* and *m:n*. This thesis proposes a catalogue of patterns, named the Process Coordination Patterns (PCPs) that describe and visualise principles for coordinating interactions among numerous related process instances at run-time. The PCPs intend to provide a framework that supports process designers in modelling business scenarios involving complex process interactions. The PCPs may be further used as guidance for evaluating the degree to which existing approaches capture more complex process interactions.

1.2. Contribution

Over the past decade the broad range of PrMS supporting the design and enactment of business processes called for formal conceptual foundations. The aim was to use those foundations as a reference point for the evaluation and improvement of PrMS, in both the commercial and research domains [33, 61]. In 1999, the Workflow Patterns Initiative was started in an effort to identify a conceptual foundation for assessing the strengths and weaknesses of existing approaches for process specification and implementation. An empirical method for identifying requirements for PrMS on a recurrent basis has been applied since then, documenting them in the form of patterns. This pattern-based approach represents means for describing core conceptual constructs inherent in process technology. Patterns can be used to evaluate the suitability of a wide variety of PrMS [43]. Whilst the first pattern catalogue, which was published in 2003 only focused on the control-flow perspective of business processes [57], over time new pattern catalogues focusing on other core perspectives have been released. At present, the prevailing perspectives encompass the data-[15, 42, 44], flexibility-[33, 37, 59], interaction-[3, 4, 55], resource-[17, 42] and time-perspective [26] among others.

However, the existing pattern catalogues are not paradigm-independent. They mainly describe a series of constructs and characteristics and offer solutions to problems frequently encountered in existing PrMS based on monolithic process models. This poses a problem especially when complex business scenarios involving multiple interactions

among related process instances are to be modelled by taking the existing patterns as guidance. The existing pattern catalogues do not fully address relationships of the type *1:n*, *m:1* and *m:n* that interacting process instances may have with one another.

On the one hand, the pattern catalogues for the interaction-perspective acknowledge the fact that the number of interacting process instances pose a significant challenge for process modelling [3, 4, 55]. On the other hand, they still primarily focus on situations with just a few isolated process instances which interact with one another via message exchanges. However, the real challenge is to cope with scenarios where the number of interacting process instances in different relationships begin to scale up. Hence, business scenarios involving multiple interacting process instances may not be modelled in detail with the existing pattern catalogues for the interaction-perspective. An example is a recruitment process, where the invitation for a job interview depends on the number of positive reviews assigned to an application.

A comprehensive catalogue of patterns describing interactions and relationship types between several process instances at a comprehensive level and which abstracts from specific paradigms is required. However, such a pattern catalogue describing multiple process interactions in an abstract and simple manner has not been provided so far. On that account, the set of patterns provided in this thesis, the PCPs, aim to serve as a generic basis for evaluating the extent to which existing BPM approaches can be used for modelling interactions among process instances that go beyond a *1:1* relationship. The PCPs thus consider *1:n*, *m:1* and *m:n* relationships. The proposed PCPs may be used to spot the difficulties that existing BPM approaches have when dealing with complex process interaction. Based on the findings, process designers could be able to modify or extend the approaches, such that these provide a more complete support for modelling complex process interactions. It is not claimed that the present pattern catalogue gathering seven patterns is complete. The PCPs are the result of an explorative work as they have been discovered by looking at business scenarios involving numerous interacting processes. In the future it is likely that more rare patterns may be found and added to the current catalogue.

1.3. Outline

The remainder of this thesis is structured as follows. Chapter 2 describes the fundamentals for understanding the context of the PCPs; in particular the notions of processes, concurrency, relations and interactions are presented. Furthermore, this chapter comprises the pattern format used and the graphical notation for describing the PCPs. Chapter 3 describes the PCP catalogue in detail. Chapter 4 evaluates how two methodologically selected modelling languages based on different paradigms support the PCPs. Chapter 5 discusses related work, before concluding the paper with a summary and an outlook in Chapter 6.

2
Fundamentals

This chapter describes the basic concepts and notions needed for understanding the context in which PCPs are used. The notions of processes, concurrency, relations and interactions are presented. Based on the described concepts and notions, comprehensive concepts related to multiple process interactions are introduced. Furthermore, the format used to describe the patterns and the graphical notation employed to illustrate them are explained in this chapter.

2.1. Pattern Context

This thesis belongs to the BPM field of techniques and tools to support the design and enactment of operational business processes. While modelling and executing business processes, it is important to realise that there is a great variety in which related processes may interact with one another. At specific points in time some processes engage in single interactions, while others are interlinked in multiple interactions. The design of scenarios involving numerous process interactions in a comprehensive manner requires looking at process interactions at a comprehensive level. The following running example may be used throughout this section to delineate the basic concepts of multiple related and interacting processes.

Running example: *In a logistics process a customer order placed at an online shop may contain several products. Based on the availability of the products, the customer order may be split into multiple packages. The packages related to one customer order in turn may be distributed with multiple delivery tours at various points in time. While one package has been already delivered, another one belonging to the same customer order may be prepared for delivery. Typically, each delivery tour comprises several packages from different customer orders while one customer order is related to several delivery tours. This relation between customer orders and delivery tours constitutes a many-to-many (m:n) relationship. On each delivery tour, the packages are delivered one after the other. In case that a package cannot be delivered, it is rescheduled for another delivery tour. If once again the package cannot be delivered, it is then returned to the shop as undeliverable. Once all packages corresponding to a customer order have been processed, the order is billed to the customer*.

From the running example it may be deduced that in such scenarios several instantiated processes are interrelated through different types of relationships. Furthermore, process instances that have been instantiated from the same process may be executed concurrently. Moreover, process instances interact with other ones at certain points in time based on specific conditions. Those statements shall be explained in-depth by abstracting the basic elements and concepts of multiple process interactions in the subsections hereafter. The catalogue of PCPs is compiled based on these foundations.

2.1.1. Basic Concepts of Multiple Process Interactions

2.1.1.1. Process Notion

A generally accepted technique on how to model business processes does not exist. As a result, the BPM field is filled with different paradigms for the modelling of business processes such as the activity-centric paradigm, the case-handling paradigm and the more recent data-centric paradigm [43]. Nonetheless, when looking at all paradigms together a common notion of what constitutes a business process can be abstracted. All paradigms have in common how a business process is perceived: *the fulfilment of particular steps towards a specific goal*.

This common notion is the motivation for the PCPs to describe business processes in a generic manner. In the context of the PCPs it is postulated that a business process may be regarded as a composite system of distinct individual types of processes, so-called *process types.* These are instantiated and interact with one another at certain points in time in order to accomplish a common goal. In the logistics process described in the running example, three different process types can be distinguished: customer order, package and delivery tour.

Within the notion of what constitutes a process, a distinction between *process type* and *process instance* has to be made. While process types represent entities at design-time, process instances represent entities of instantiated process types at run-time.

Each process type may follow a *lifecycle* describing the process's behaviour. The lifecycle of a process type may be seen as a *state-based-view* describing the execution states of the process type from its creation to its disposal. The notion of *state-based-views* has been borrowed from the object-aware process paradigm [47], because it allows for a more generalised view on processes. In particular, the idea of state-based-views fosters an abstract perception of processes such that the PCPs may project processes in a paradigm-independent manner. In the following, only the concepts of the state-based-view that are relevant for the understanding of this thesis are mentioned; further details can be found in [1, 25, 47].

Every process type has its own individual lifecycle such that state-based-views of process types differ in states and lengths of the lifecycle. A state-based-view may be illustrated as a directed graph consisting of nodes and directed edges connecting the nodes. The

nodes represent the states of the process's lifecycle, while the directed edges represent *transitions* indicating that the process may pass from one state to another but not vice versa. Thus, transitions put states into an unidirectional sequence. There is one *start state* S_S , an arbitrary number of *intermediate states* S_I and at least one *end state* S_E. The directed structure of the lifecycle implies that only one state can be executed at a specific point in time, i.e. only one state may be active. Consequently, branching transitions in the state-based-view thereby have a XOR-semantic. The return to a previous state within the lifecycle may be exceptionally enabled by *backwards transitions*, which may be pictured as reversed edges. While the start state S_S is the only one that is not the destination of any transition, the end state S_E is the only one that is not the source of any transition.

In the following, the running example is used to illustrate the described concept of the state-based-view. An exemplary illustration of the state-based-view of the process type *package* derived from the process description is shown in Figure 2.1. In this illustration none of the states is active such that the state-based-view generally describes the lifecycle of the process type *package* at design-time. The lifecycle of a package starts with the package being *packed*, then the package is *loaded* into the delivery truck. In the next state the *delivery* of the package takes place and the package is either *delivered,* or in case that a delivery fails, the package is *rescheduled* for delivery. If a rescheduled delivery fails again, the package is finally returned to the store as *undelivered*. After a package has been either *delivered* or returned as *undelivered* the lifecycle of the package ends with the package being marked as *processed*.

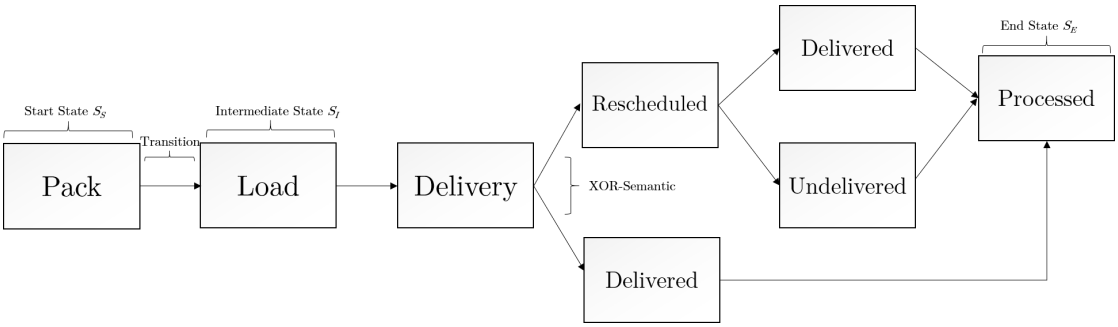


Figure 2.1.: State-based-view of the Process Type $Package$ at Design-time

2.1.1.2. Concurrency / Parallelism

At run-time state-based-views capture the behavior of individual process instances. A process type can be instantiated multiple times, such that several single process instances from the same process type coexist.

Going back to the running example, a customer order may be split into several packages, such that at run-time multiple process instances of the type *package* may be created.

However, process instances are not necessarily instantiated all at once but at different points in time. For example, process instances of the type *package* are instantiated based on the availability of the products in stock. In this case a package may be packed with those products and loaded into the delivery truck. In the meantime, products that are not in stock are ordered and once they arrive the next package might be prepared. Hence, process instances that have been instantiated from the same process type at either the same or different points in time may be situated at different states of their lifecycle. Process instances travel through their corresponding lifecycle at their own pace mostly independently from other process instances. In other words, process instances may run asynchronously and concurrently.

The notion of concurrent process instances at run-time is pictured in Figure 2.2 where the active states of the process instances are marked. While $package_1$ is situated in the state *processed* of its lifecycle, $package_2$ is in the state *rescheduled*.

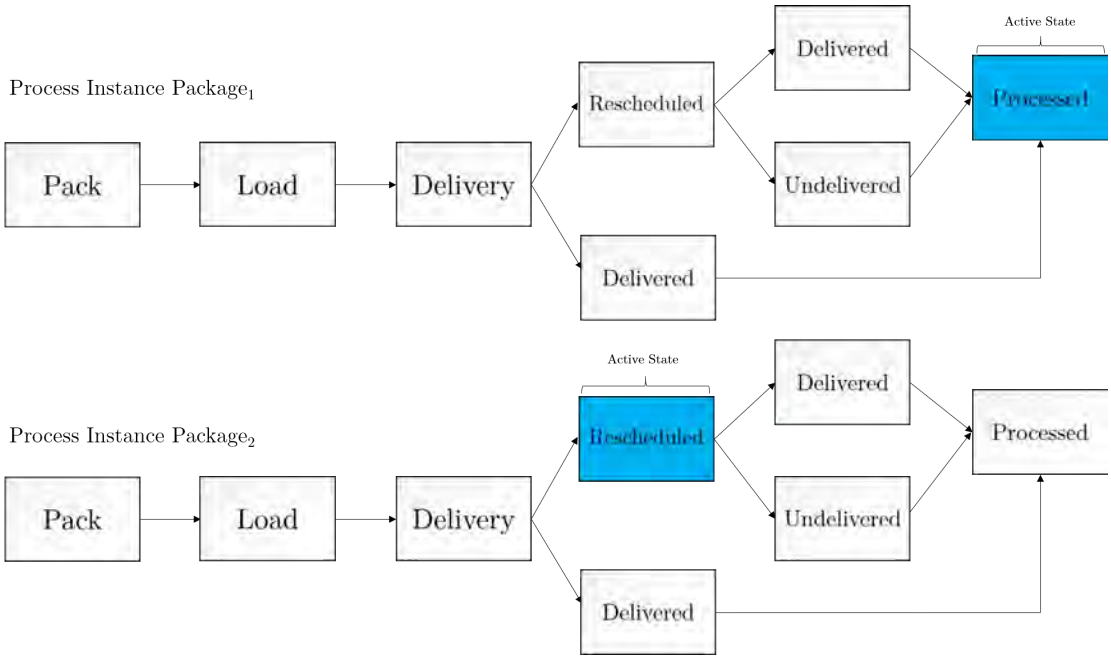


Figure 2.2.: Concurrent Process Instances $Package_1$ and $Package_2$ at Run-time

While in general terms *concurrent* and *in parallel* are mostly used as substitutes in the general literature, in the Computer Science domain the meaning differs. The term *concurrent* relates to the alternating execution between two threads such that they advance independently of each other on the same processor. Thread one is first executed and then suspended; thereafter thread two is executed, suspended and so on. The term *in parallel* refers to two threads, both being executed simultaneously on different processors - that is literally at the same time [52]. In the context of this

thesis, this distinction is mostly unnecessary, so for reasons of simplicity only the term concurrent is used, summarising concurrent and parallel execution.

2.1.1.3. Relations

Between different process types different types of relations may exist. A relation can be of the type *1:1*, *1:n*, *m:1* or *m:n*. At design-time, relations indicate that the involved process types have something to do with each other. This means that at run-time, process instances that have been instantiated from related process types may interact with each other at a specific point in time. In the running example several relations between the involved process types can be found. One order may be split into several packages, which poses a one-to-many relation. Several packages may be distributed with one delivery tour, which constitutes a many-to-one relation. Multiple orders from distinct customers may be distributed with distinct delivery tours, which represents a many-to-many relation.

In order to look at process relations in detail, relationships between process types can be further divided into *incoming* and *outgoing relations*. Based on this, in the context of the PCPs, it is set that relations between process types are directed. Depending on incoming and outgoing relations, process types are differentiated into *source process types* and *target process types* at design-time. Source process types have outgoing relations while target process types have incoming relations. By analogy, at run-time related process instances are differentiated into *source process instances*, which have outgoing relations and *target process instances,* which have incoming relations.

At design-time, process types may not only be related to one further process type but to several process types where each relation can be of a different type. Consequently, based on the context, one process type can be a target process type in one relation and a source process type in another relation. The same applies for process instances at run-time.

The relations set at design-time for the process types of the running example are illustrated in Figure 2.3. The process type *order* has a one-to-many relation to the process type *package* where the former is the source process type and the later is the target process type. The relation between the process types is pictured as a directed dashed edge with the type of relation indicated above. In turn, the process type *package* has a many-to-one relation to the process type *delivery tour*. In this relation the process type *package* is a source process type while *delivery tour* is the target process type.

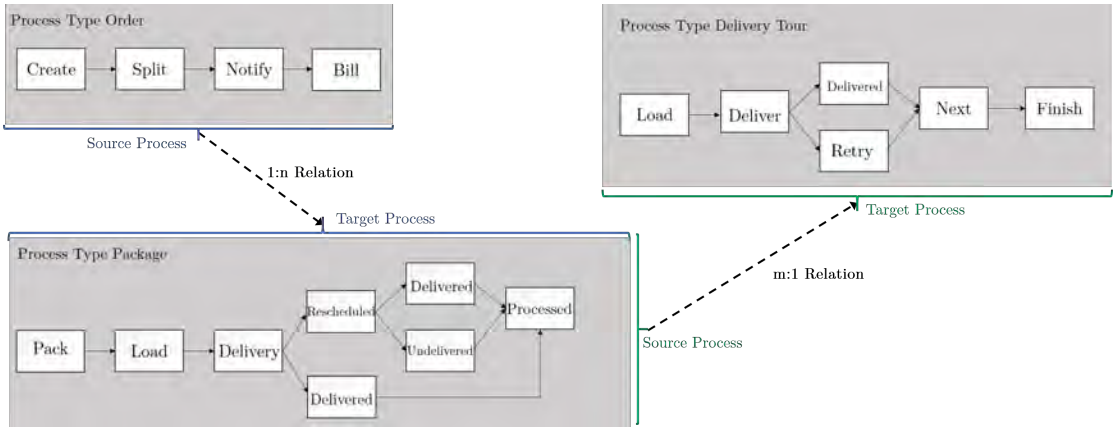


Figure 2.3.: Relations between Process Types at Design-time

Different process types and the multiple relations between them may generate large *process structures*, which can become very complex when enacting them at run-time. Commonly, multiple source process instances may be related to multiple target process instances. Knowing which source process instances are related to which target process instances at run-time is relevant for handling process interactions adequately.

For this purpose, the PCPs are used for identifying and managing the relations that may exist between process types in business scenarios. This is achieved through a parameter which is pre-defined at design-time. This parameter specifies the number of source process instances that may be related to a number of target process instances at run-time, this degree of relation being referred to as *cardinality*.

In the context of the PCPs, a relationship represents by default a *m:n* relationship between the related source and target process types where the cardinality *m* is assigned to the source process type and the cardinality *n* to the target process type. Furthermore, each of the cardinalities *m* and *n* may be restricted by a *lower* and an *upper bound* of the form m_{lower}, m_{upper}, n_{lower}, n_{upper} such that the number of process instances in relationship with each other can be restricted at run-time. It applies

$$(m_{lower} \leq m_{\text{upper}}), (n_{lower} \leq n_{upper}) \quad .$$

By setting cardinality restrictions, relationships may also be reduced to a *1:n*, *m:1* or *1:1* relationship. Cardinality restrictions imply that related process instances may only be created within the values assigned to the bounds.

In this context, the lower bound m_{lower} specifies the minimum number of source process instances that can be related to target process instances, that lie within the range $n_{\text{lower}}..n_{\text{upper}}$ assigned to the target process instances. The upper bound m_{upper} specifies the maximum number of source process instances that can be related to target process instances, that lie within the range $n_{\text{lower}}..n_{\text{upper}}$ assigned to the target process instances.

Based on the stated concepts concerning the assignment of cardinalities, the following notation may be used for the PCPs:

$$(m_{\text{lower}}..m_{\text{upper}}) : (n_{\text{lower}}..n_{\text{upper}})$$

In the following, the running example is used for describing the established concepts. Within the logistics process it may be determined that for economic reasons the delivery truck driven in a delivery tour must load at least 20 medium size packages out of the maximal capacity of 50 medium size packages before the delivery process is started. In this scenario the cardinality *m* assigned to the source process type *package* and the cardinality *n* assigned to the target process type *delivery tour* at design-time is written as (20..50) : 1, where m_{lower} = 20, m_{upper} = 50, $n_{lower} = n_{\text{upper}}$=1. This relation specifies that at run-time not less than 20 packages and up to 50 packages may be loaded into one delivery truck. The defined upper bound m_{upper} = 50 assigned to the process type *package* states that further process instances of the type *package* may be packed and loaded into the delivery truck as long as the upper bound is not exceeded. For example if 48 process instances of the type *package* are already loaded into the delivery truck, up to two additional packages could be packed and loaded into the delivery truck before the delivery process is started.

2.1.1.4. Interactions

As it has been stated in the sub-subsection *Concurrency / Parallelism*, at run-time process instances are in principle independent from one another, each one travelling through its corresponding lifecycle at its own pace. However, interactions among related process instances are necessary at certain points in time. Therefore, *directed relations* among process types are already established at design-time. Directed relations indicate that source process instances may interact with related target process instances at run-time.

In a general way, interactions stand for process dependencies, indicating that the execution of certain process types may depend on the execution status of other process types. While commonly each process instance runs independently from others, at a specific point in time a *number* of process instances of the same type needs to be synchronised before related process instances can be executed.

Going back to the running example, a typical interaction is that an order cannot be billed to a customer unless *all packages* related to that specific order have been processed. For example, if an order is split into three packages and $package_1$ is in state *processed*, while $package_2$ is in state *delivery* and $package_3$ in state *rescheduled*, the bill may be issued only when *all* three packages are synchronised as *processed.*

Inherently, interactions between related process instances imply that the synchronised execution state of a number of source process instances needs to be *coordinated*[1] with the execution state of related target process instances for the overall business process to advance. In conclusion, the execution of target process instances depends on the

[1] The name assigned to the PCPs is derived from this concept. The term *interaction* is intentionally not part of the name assigned to the PCPs in order to stand out from the patterns in the interaction-perspective already proposed by other authors.

execution status of related source process instances. However, after interactions have taken place, source and target process instances may continue through their lifecycles concurrently and asynchronously to each other.

The number of source process instances that are synchronised into a specific execution state represent a threshold value which needs to be reached such that related target process instances can be executed. In the context of the PCPs, this threshold value is denominated as ω. Likewise the cardinality restrictions m_{lower} and m_{upper}, the threshold value ω is assigned to a source process type. Due to the dynamics of business processes, where instances may be created or deleted at any point in time, ω may be assigned at both, design- and / or run-time. For ω applies that it must lie between the lower and upper bounds m_{lower}, m_{upper} of the source process type, such that the following equation holds:

$$(m_{\text{lower}} \leq \omega \leq m_{\text{upper}})$$

Bringing together the notions of cardinality and threshold value assignment, follows:

$$(m_{lower} \leq \omega \leq m_{\text{upper}}), (n_{lower} \leq n_{upper})$$

Interactions among related process instances can be illustrated using state-based-views. In Figure 2.4 the example described above is exemplarily illustrated.

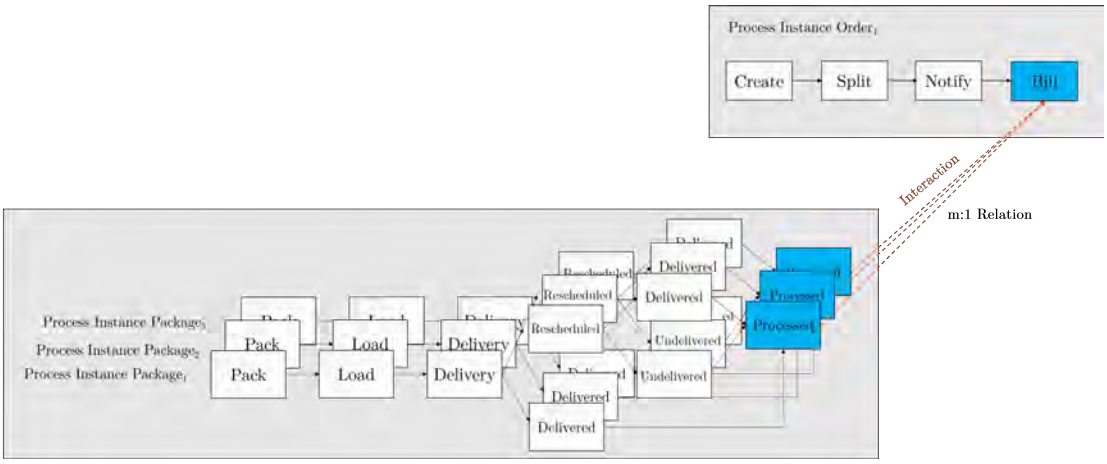


Figure 2.4.: Interactions between Process Instances at Run-time

It can be seen that three process instances of the type *package* that are related to one process instance of the type *order* are synchronised into the state *processed.* The synchronised states of the process instances are visualised with coloured markings (blue). Based on the execution status of the source process instances $package_{1,2,3}$, the target process instance $order_1$ can proceed into the state *bill*. This is visualised with the coloured marking (blue) of the state *bill*. Dashed edges going out from the execution states *processed* of the process instances $package_{1,2,3}$ into the execution state *bill* of the target process instance $order_1$ visualise the resulting interaction.

Concerning the cardinalities, in this scenario the cardinality *m* assigned to the source process type *package* and the cardinality *n* assigned to the target process type *order* are written as (1..n) : 1, where $m_{\text{lower}} = 1$, $m_{upper} = n$, $n_{lower} = n_{\text{upper}} = 1$. This relation specifies that at run-time at least *one package* and up to *n packages* are related to one *order*.

Regarding the threshold value ω, the condition "once all packages are processed..." represents the number of source process instances of the type *package* that need to be synchronised such that a related target process instance of the type *order* may be billed. In this case it applies $\omega = n$, such that $\omega = m_{upper} = n$ with *n* specifying the maximal number of *packages* that can be related to an *order*.

2.1.2. Comprehensive Concepts of Multiple Process Interactions

In the PCPs context, a set of process instances is denoted as A_n, where A is the name of a specific process type and n is a unique identifier assigned to individual process instances. In order to easily differentiate between source and target process types, in this thesis, the characters A to F are assigned to source process types, while the characters U to Z are used for target process types. For example, a source process type *A* may have instances A_1, A_2, A_3..A_n which may be related to target process instances U_1, U_2, U_3..U_n.

For reasons of understandability, until now the presented concepts have only taken into account interactions between the instances of two related process types, i.e. one source process type and one target process type. However, the discovered PCPs reveal a context where *multiple interdependent process types* need to interact in order to reach a specific business goal. More precisely, one source process type may be related to numerous target process types and also numerous source process types may be related to one common target process type. In those cases two scenarios arise: *a)* multiple instances of one source process type may interact with multiple instances of several related target process types and *b)* multiple instances of several source process types may interact with multiple instances of one common target process type.

The PCPs take into account multiple relations and enable the assignment of different cardinalities and threshold values among process types. To be more precise, in the scenario *a)* where one source process type is related to several target process types, different cardinalities *n* may be assigned to the target process types. However, the cardinality *m* and threshold value ω assigned to the source process type must remain the same. This is due to the fact that, the execution of all target process types depends on the execution status of the related source process type.

Conversely, in the scenario with several source process types related to one target process type, the cardinality *m* as well as the threshold value ω assigned to source process types may vary for each one. However, the cardinality *n* assigned to the common target process type must remain the same. That follows from the fact that, the execution

of the common target process type depends on the execution status of all related source process types.

On a related note, when referring to the threshold value ω assigned to source process types, the following naming convention applies in the context of the PCPs: ω^{A}, where superscript *A* indicates the source process type to which the threshold value ω is assigned.

To clarify the described concepts, the following scenario may be used as an example. Two medical tests *A* and *B* exhibiting a probability of errors, may be performed several times within a month. Depending on the number of positive results of both tests a reliable diagnosis *Z* can be made. The medical tests *A* and *B* may be performed up to three times each. Medical test *A* requires a minimum of one positive result and test *B* a minimum of two positive results before a disease can be surely confirmed. The described example is illustrated in Figure 2.5.

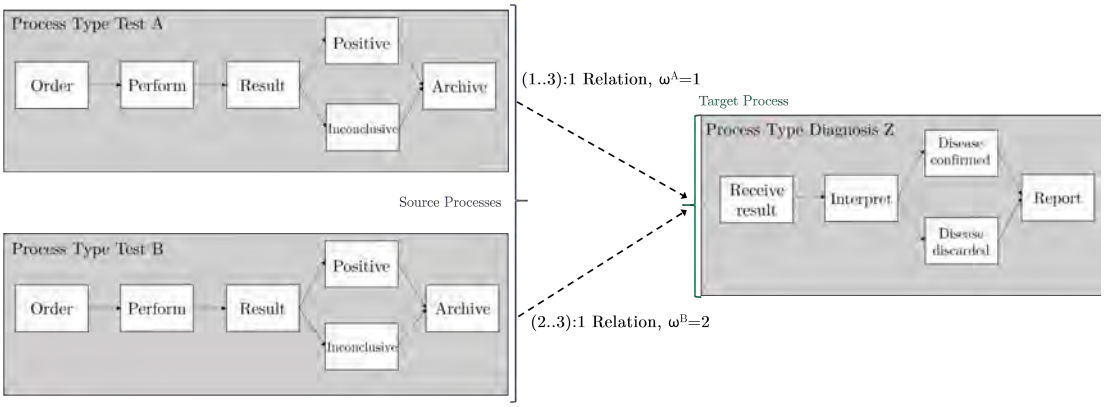


Figure 2.5.: Example Scenario of Multiple Relations between Process Types

In this scenario the source process types *A* and *B* are related to one common target process type *Z*. Hence, in general different cardinalities m_i, m_j may be distinguished and assigned to each source process type. Thus, the different cardinalities for processes *A* and *B* can be denoted as m^{A} and m^{B}. The cardinality n^{Z} may be assigned to the target process type *Z*, with which the source process types *A* and *B* are both related. Moreover, two threshold values ω^{A} and ω^{B} may be distinguished and assigned to the source process types *A* and *B*.

The cardinality assigned to the source process type *A* and the target process type *Z* is written as (1..3) : 1, where ${m^A}_{\mathrm{lower}} = 1$, ${m^A}_{\mathrm{upper}} = 3$, ${n^Z}_{\mathrm{lower}} = {n^Z}_{\mathrm{upper}} = 1$ and the threshold value $\omega^{\mathrm{A}} = 1$. This notation specifies that at run-time at least one test and at the maximum three tests of the type *A* may be related to one diagnosis Z. The threshold value $\omega^{\mathrm{A}} = 1$ represents the number of tests *A* that must be *positive* for a reliable diagnosis *Z* to be made.

The cardinality assigned to the source process type *B* and the target process type *Z* is written as (2..3) : 1, where $m^{B}{}_{\text{lower}} = 2$, $m^{B}{}_{\text{upper}} = 3$, $n^{Z}{}_{\text{lower}} = n^{Z}{}_{\text{upper}} = 1$ and the threshold value $\omega^{\text{B}} = 2$. This notation specifies that at run-time at least two and up to three test of the type *B* may be related to one diagnosis *Z*. The threshold value $\omega^{\text{B}} = 2$ represents the number of tests *B* that must be *positive* for a reliable diagnosis *Z* to be made.

However, only if both threshold values ω^{A} and ω^{B} have been reached, a conclusive diagnosis *Z* can be made. In other words, once a determined number of source process instances A_n and B_n reach a specific execution status, a common target process instance Z_1 can be executed.

In this context, the question may arise as to what occurs with unsynchronised source process instances that remain after the threshold value ω has already been reached. This scenario arises when the threshold value ω does not involve the whole set of existing source process instances but only a subset such that $\omega < m_{upper}$. Whether the remaining process instances might be discarded or remain with no further effect is left as a *design choice* to the process designer.

Going back to the scenario with the two medical tests *A* and *B*, an example illustrated in Figure 2.6. is used to clarify the described approach.

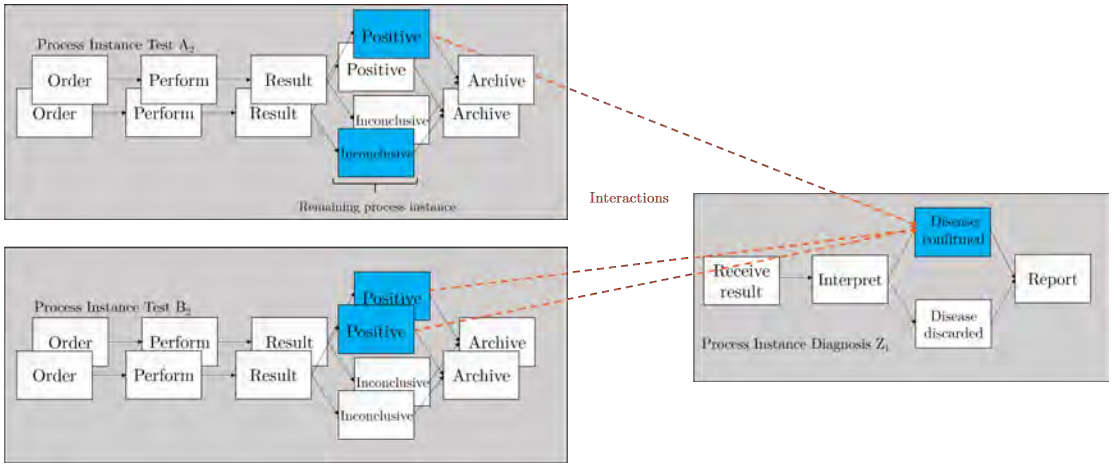


Figure 2.6.: Example Scenario of Multiple Process Interactions and Remaining Process Instances

It may be assumed that at run-time the performed medical test A_1 is evaluated as *inconclusive* while two performed medical tests B_1 and B_2 are evaluated as *positive*. In the event that a second test A_2 is performed and evaluated as *positive*, a conclusive diagnosis *Z* could be made. In this scenario medical test A_1 is not part of the resulting interactions between process type *A* and process type *B* with the common process type *Z*. Whether medical test A_1 is discarded or archived is thus left as a design choice. In the illustrated example remaining process instances are archived.

As mentioned throughout this sub-subsection the number of source and target process types related to one another may vary. The specific combination of related source and target process types constitutes a concrete pattern. In this context, the default quantity of process types that constitute each pattern is defined with the parameters P_{Source} and P_{Target}. P_{Source} circumscribes the minimum number of source process types, while P_{Target} circumscribes the minimum number of target process types necessary for forming the pattern.

2.2. Pattern Format

Patterns provide descriptions for frequently recurring problems that PrMS are facing. A prerequisite for the effective application of patterns is documenting them using a systematic approach. To ensure a comparable and precise description, the Process Coordination Patterns proposed in this thesis are documented uniformly, based on the pattern standard template proposed by Gamma et al. in [18]. The pattern format used in this thesis contains the following sections:

- **Name:** A succinct name describing the essence of the pattern.
- **Essentials:** The rationale and intent of the pattern.
- **Prerequisite:** The specific combination of related source and target process types that constitute the pattern.
- **Description:** A short summary of the pattern's functionality.
- **Structure:** A graphical representation of the pattern's functionality using a generic and language-independent notation.
- **Example:** An illustrative example of the pattern's applicability.
- **Remarks:** Issues potentially encountered when implementing or using the pattern.

The sections *description*, *structure* and *example* are used for depicting *1:1*, *1:n* / *m:1* and *m:n* relationships for each pattern, so that the concept of different process interactions can be understood.

2.3. Graphical Representation of PCPs

The graphical representation of the PCPs is based on simple graphic elements and constructs. The representation aims to capture the different types of relationships that interacting process instances may have with one another. Relationships may be of the form 1:1, 1:n / m:1 and m:n. The graphic elements of the PCPs are shown in Figure 2.7.

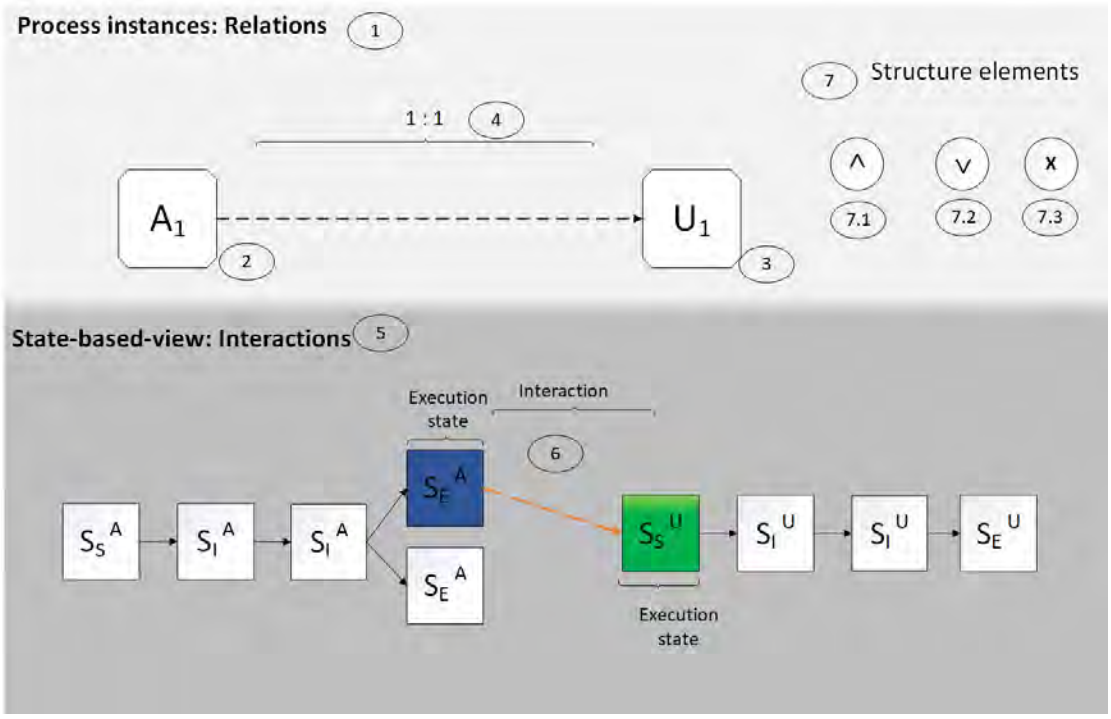


Figure 2.7.: Graphic Elements for Describing PCPs

On the upper level of the figure *process instances* and their relationships are illustrated (1).

Process instances are visualised as rectangles with cut-off corners placed one behind the other, referencing the process type from which they have been instantiated. Process instances have a unique identifier *n*.

For example A_1 represents the first *source process instance* (2) that has been instantiated from the source process type *A*. Accordingly, U_1 represents the first *target process instance* (3) that has been instantiated from the target process type *U*.

Relationships between instances of a process with instances of another process are expressed by directed dashed edges between source and target process instances with *the type of relationship* (4) indicated above the dashed edges. As seen in Figure 2.7, the source process instance A_1 and the target process instance U_1 have a *1:1* relationship.

On the lower level of the figure the state-based-views of interacting process instances are illustrated (5).

Interactions (6) between related source and target process instances are visualised by directed dashed edges between specific execution-states of their state-based-views. As shown in the figure, the interaction between A_1 and U_1 is based on the end state ${S_E}^A$ of the source process instance A_1 which is coordinated with the start state ${S_S}^U$ of the related target process instance U_1.

Three types of *interaction indicators* (7) are used as visual aids to indicate the different types of interactions that may exist among related process instances at run-time.

For the first type of interaction indicator (7.1) two cases are distinguished. The circle with a $\wedge$ sign is used for indicating that: *a)* instances of one source process type may concurrently interact with instances of all target process types, *aa)* instances of all source process types may concurrently interact with instances of a common target process type. The circle with a $\wedge$ indicates "AND" semantics.

For the second type of interaction indicator (7.2) also two cases are distinguished. The circle with a $\vee$ is used to indicate variations of concurrent process interactions: *b)* instances of one source process type may concurrently interact with instances of one to multiple target process types, *bb)* instances of one to multiple source process types may concurrently interact with instances of a common target process type. For the case *b)* applies that out of the entire set of target process types, instances of at least one target process type are part of the interaction. For the case *bb)* applies that out of the entire set of source process types, instances of at least one source process type are part of the interaction. The circle with a $\vee$ indicates "OR" semantics.

For the third type of interaction indicator (7.3) only one case applies. The circle with an X is employed to indicate that: *c)* instances of one source process type may interact with instances of not more than one target process type out of the entire set. Once instances of a source process type start interacting with instances of a target process type, the instances of the remaining target process types are excluded from interacting. Business scenarios exhibiting another type of interaction than the one described here could not be found so far. The circle with an X indicates "XOR" semantics.

3

Process Coordination Patterns

This chapter provides an overview of the Process Coordination Patterns that have been found during the process of pattern discovery in the frame of this thesis. First, a catalogue describing the PCPs in detail is presented. Thereafter, insights acquired during the identification and analysis of the PCPs are discussed.

3.1. PCP Catalogue

The PCP catalogue is a collection of currently seven patterns describing different types of interactions that may exist among process instances that have relationships with one another of the type *1:1*, *1:n* / *m:1* and *m:n*. In the PCP catalogue the interactions between instances of different process types are explicitly modelled.

The PCPs are an abstraction of process interactions which repeatedly occur in business scenarios, involving multiple related and interacting processes. All PCPs are based on fundamental principles. In order to avoid unnecessary repetition when describing each pattern, these principles are listed hereinafter:

- At design-time a relationship represents by default a *m:n* relationship between the source and target process types where the *cardinality m* is assigned to the source process type and the *cardinality n* to the target process type. *Cardinalities* specify the number of source process instances A_1, A_2, A_3..A_n that can be related to a number of target process instances U_1, U_2, U_3..U_n at run-time. Each of the cardinalities *m* and *n* may be restricted by a *lower* and an *upper bound* of the form m_{lower}, m_{upper}, n_{lower}, n_{upper} such that based on the values assigned to the lower and upper bounds of *m* and *n,* the default *m:n* relationship can be modified to a *1:1, 1:n or m:1* relation.
- The threshold value ω specifies the number of source process instances A_n that need to reach a specific execution state of their lifecycle such that related target process instances U_n can be executed. The threshold value ω may be assigned to a source process type at both, design- and / or run-time. For ω applies that it must lie between the lower and upper bounds m_{lower}, m_{upper} of the source process type.
- From the principles of cardinality and threshold value assignment follows:

$$(m_{lower} \leq \omega \leq m_{\mathrm{upper}}), (n_{lower} \leq n_{upper})$$

Explanatory notes for a better understanding of the PCP catalogue

- The listed examples describing process interactions are one option of many since business processes differ from one organisation to another. Therefore, the state-based-views and the execution-states chosen for illustrating exemplary process interactions vary.
- Each of the presented patterns is described, illustrated and its existence is confirmed with an example taken from business scenarios. At first, interactions between process instances in a 1:1 relationship setting are addressed. Subsequently, interactions between process instances in a 1:n or m:1 relationship setting are described. Finally, the more complex relationship setting m:n is presented.
- For the graphical representation of each pattern, the default quantities of related process types that constitute each pattern is pictured.
- For abstraction purposes in the pattern description, in the case of a *1:n*, *m:1* and *m:n* relationship between interacting process instances, the cardinality assigned to the process types is not a specific number but remains an abstract value. For example in the case where instances of one source process type *A* interact with instances of two target process types *U* and *W* in a *1:n* relationship, the cardinality *m* with ${m^A}_{\mathrm{lower}}$=${m^A}_{\mathrm{upper}} = 1$ is assigned to *A* while the cardinality n^{U} assigned to *U* is ${n^U}_{\mathrm{lower}}$=${n^U}_{\mathrm{upper}} = n$ and the cardinality n^{W} assigned to *W* is ${n^W}_{\mathrm{lower}}$= ${n^W}_{\mathrm{upper}} = n$. In the case of interactions between instances of *A* with instances of *U* and *W* in a *m:n* relationship, the cardinality *m* with ${m^A}_{\mathrm{lower}}$=${m^A}_{\mathrm{upper}} = m$ is assigned to *A* while the cardinality n^{U} assigned to *U* is ${n^U}_{\mathrm{lower}}$=${n^U}_{\mathrm{upper}} = n$ and the cardinality n^{W} assigned to *W* is ${n^W}_{\mathrm{lower}}$=${n^W}_{\mathrm{upper}} = n$.

3.1.1. PCP 1 Simple Succession

Essentials

PCP 1 describes the basic case of process interaction. Only two process types are involved. The execution of the target process type depends on the execution status of the related source process type.

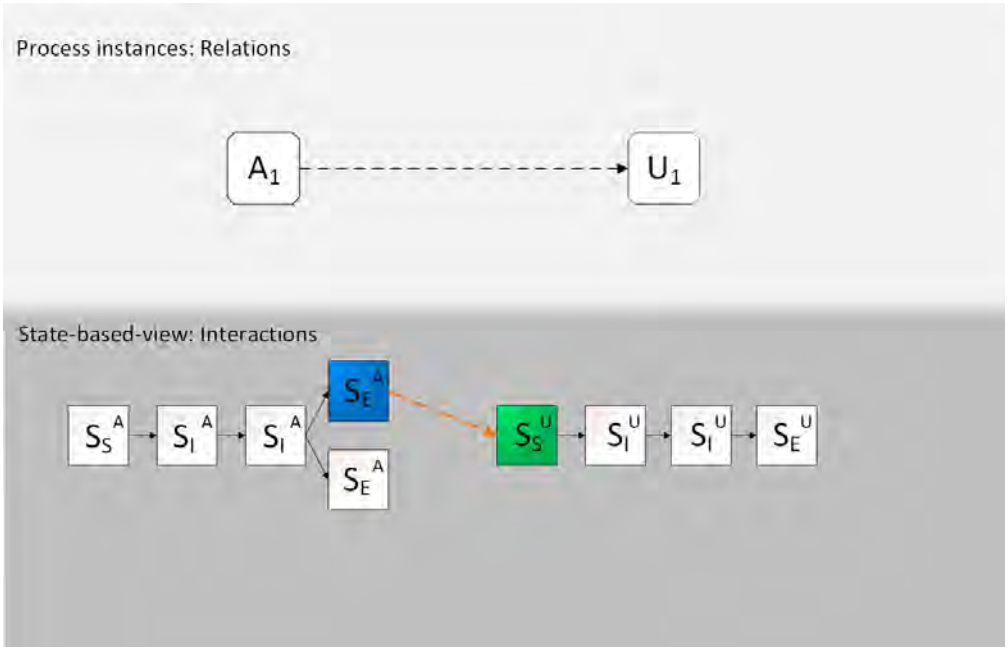
Prerequisite

P_{Source}=P_{Target}=1

Description 1:1

The execution of one single process instance U_1 depends on the execution status of another single process instance A_1.

Structure 1:1



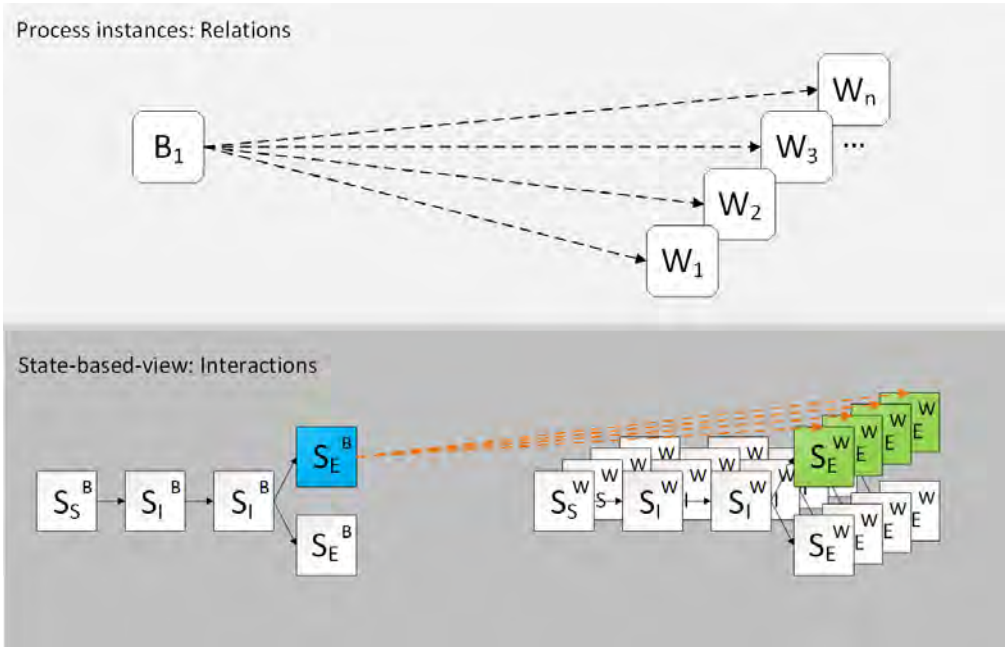
Example 1:1

A customer places an order in an online shop. Once the availability of the product in stock is verified, a notification process can be started to inform the customer.

Description 1:n

The execution of a set of process instances W_n depends on the execution status of one single process instance B_1.

Structure 1:n



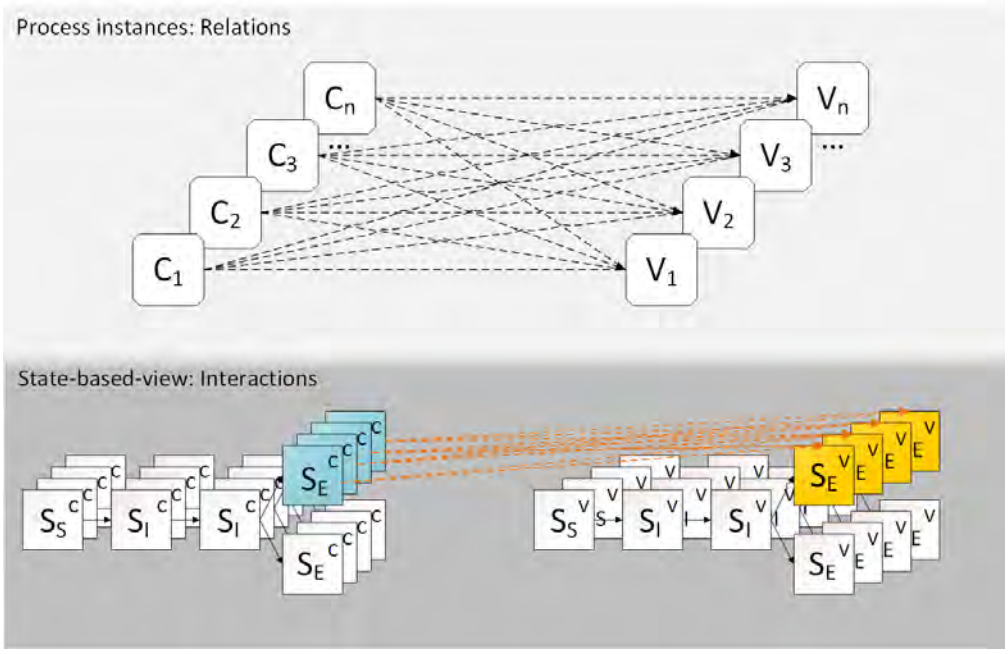
Example 1:n

A company announces a job offer. Once the position is filled in the selection process, the review process is stopped and the remaining applications are rejected [47].

Description m:n

The execution of a set of process instances V_n depends on the execution status of another set of process instances C_n.

Structure m:n



Example m:n

A customer awards a contract to an engineering company. Once the time schedules of several engineers are checked for remaining time slots, advisory support meetings with the customer can be arranged.

Remarks

No interaction indicator is needed as visual aid.

3.1.2. PCP 2 Concurrent Succession

Essentials

PCP 2 describes concurrent process interactions. One source process type is related to several target process types. The execution of all target process types depends on the execution status of the common source process type. All target process types may be executed concurrently.

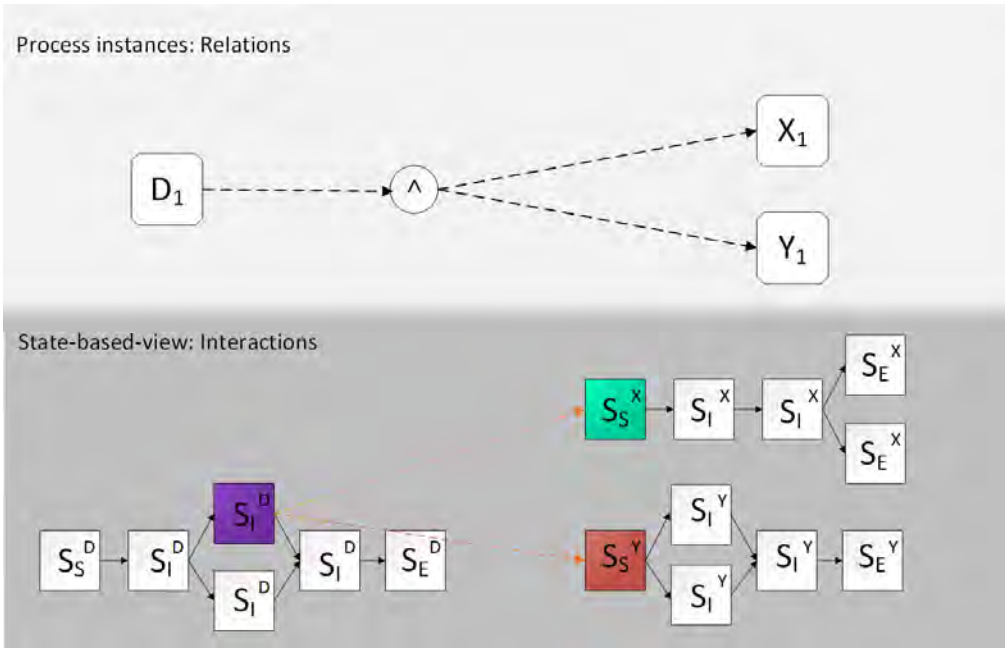
Prerequisite

P_{Source}= 1, $P_{Target}\geq$ 2

Description 1:1

The execution of two or more single process instances X_1 and Y_1 depends on the execution status of one single process instance D_1. The single process instances X_1 and Y_1 may be executed concurrently.

Structure 1:1



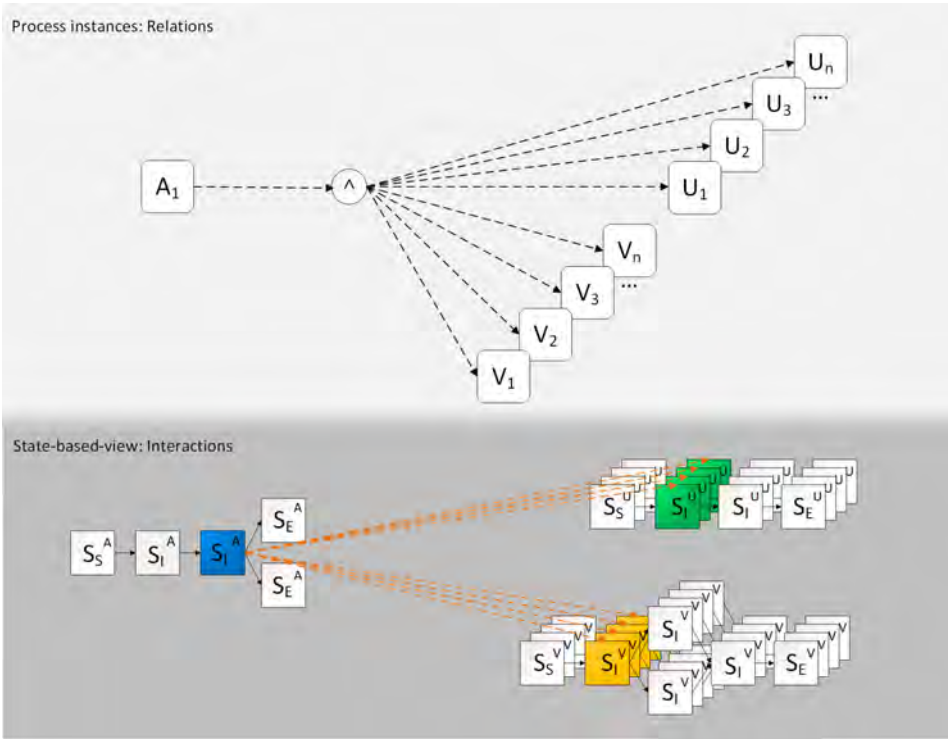
Example 1:1

A customer places an order for a product. Once the order has been processed, both the shipment and the payment processes are started.

Description 1:n

The execution of two or more sets of process instances U_n and V_n depends on the execution status of one single process instance A_1. The sets of process instances U_n and V_n may be executed concurrently.

Structure 1:n



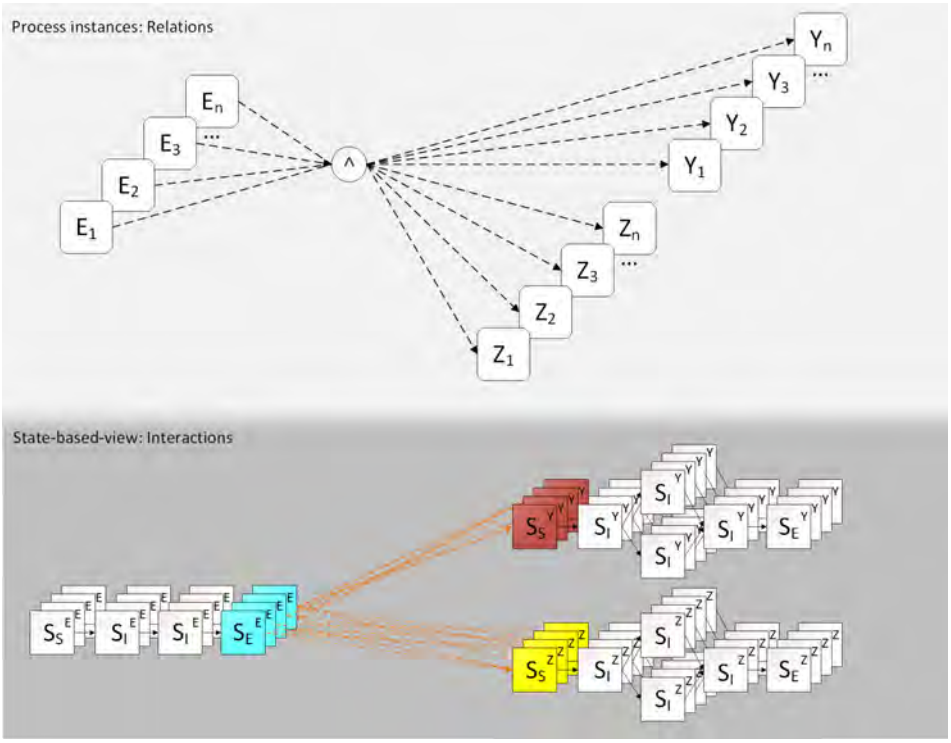
Example 1:n

A company creates a new position. After a corresponding job offer has been prepared, multiple processes are started to publish the job offer on different portals. The job offer may be advertised in social networking sites and in newspapers.

Description m:n

The execution of two or more sets of process instances Y_n and Z_n depends on the execution status of a set of process instances E_n. The sets of process instances Y_n and Z_n may be executed concurrently.

Structure m:n



Example m:n

A travel agency offers group tours. Once a minimum number of travellers are registered for a group tour, both the hotel and flight booking processes for the participants are started.

Remarks

Interaction indicator: The circle with a $\wedge$ is used as visual aid to indicate that one source process type interacts with all related target process types.

3.1.3. PCP 3 Choice

Essentials

PCP 3 describes mutually exclusive process interactions. One source process type is related to several target process types. The execution of all target process types depends on the execution status of the common source process type. Target process types are mutually exclusive, i.e. once one of them is executed, the remaining ones are permanently prevented from execution.

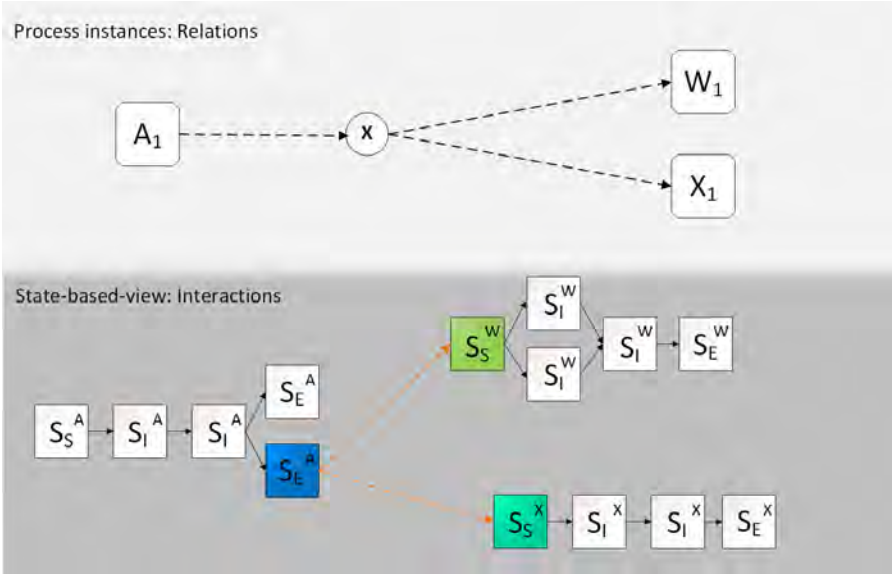
Prerequisite

P_{Source}= 1, $P_{Target} \geq$ 2

Description 1:1

The execution of two or more single process instances W_1 and X_1 depends on the execution status of one single process instance A_1. The single process instances W_1 and X_1 are mutually exclusive. Once the execution of one process instance begins, the remaining process instances are permanently prevented from being executed.

Structure 1:1



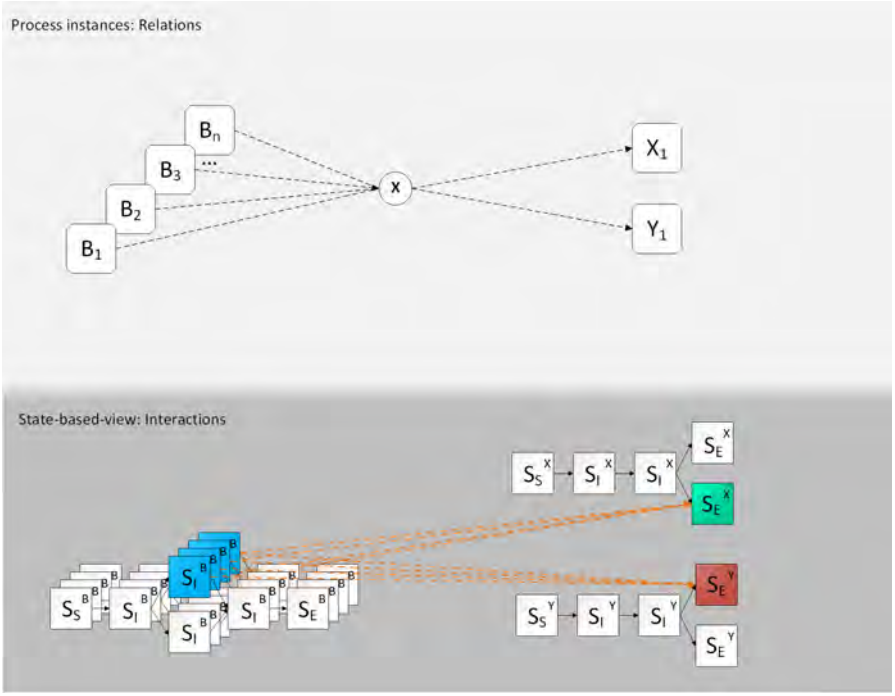
Example 1:1

A thunderstorm destroys the roof of a property; the owner is insured and requests the damage to be taken over by the insurance company. Based on the result of the insurance cover evaluation process, either a payout process granting the insured amount is approved or a rejection decision letter is send to the claimant cf. [49].

Description m:1

The execution of two or more single process instances X_1 and Y_1 depends on the execution status of a set of processes instances B_n. The single process instances X_1 and Y_1 are mutually exclusive. Once the execution of one process instance begins, the remaining process instances are permanently prevented from being executed.

Structure m:1



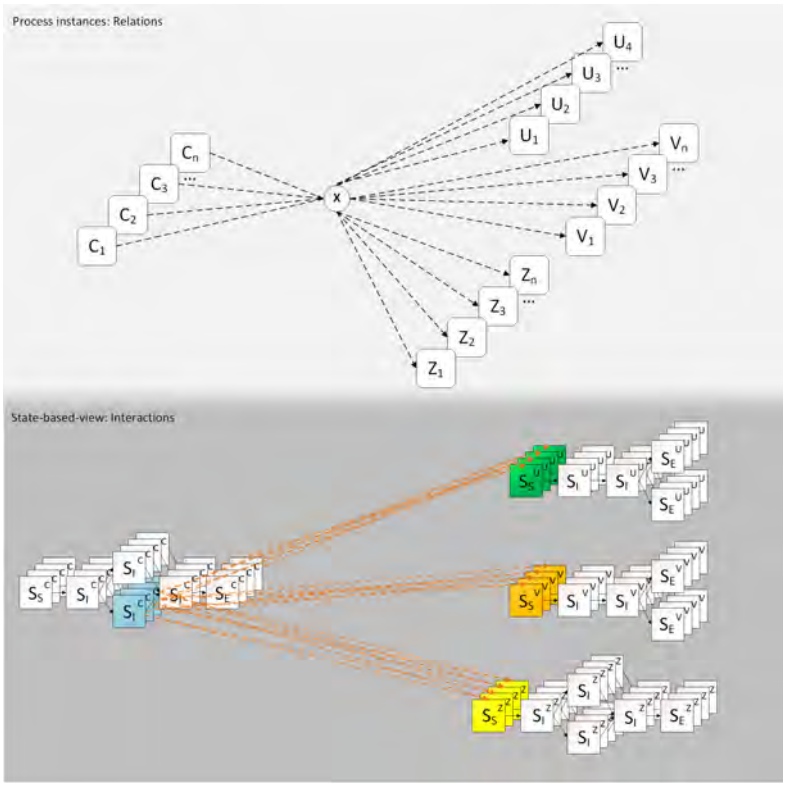
Example m:1

A company wants to slim down the product portfolio. Based on the results of an evaluation process, the company decides which variant of a product it will continue manufacturing. The production process of the less desired variant is stopped.

Description m:n

The execution of two or more sets of process instances U_n, V_n and Z_n depends on the execution status of a set of process instances C_n. The sets of process instances U_n, V_n and Z_n are mutually exclusive. Once the execution of one set of process instances begins, the remaining sets of process instances are permanently prevented from being executed.

Structure m:n



Example m:n

A business manager of a supermarket chain is responsible for the procurement of several subsidiaries. She may place several orders of the same product for different subsidiaries. The wholesaler evaluates the urgency of delivery of the orders and based on that, he decides for the most convenient means of carriage between airplain, train or truck. Multiple delivery tours for the chosen transportation are set up.

Remarks

Interaction indicator: The circle with a x is used as visual aid to indicate that one source process type interacts only with one out of the entire set of related target process types.

3.1.4. PCP 4 Coexistence

Essentials

PCP 4 describes a special case of concurrent process interactions. One source process type is related to several target process types. The execution of all target process types depends on the execution status of the related common source process type. Target process types are non- mutually exclusive. At least one target process type is executed. Subsets of the set of target process types may be executed concurrently.

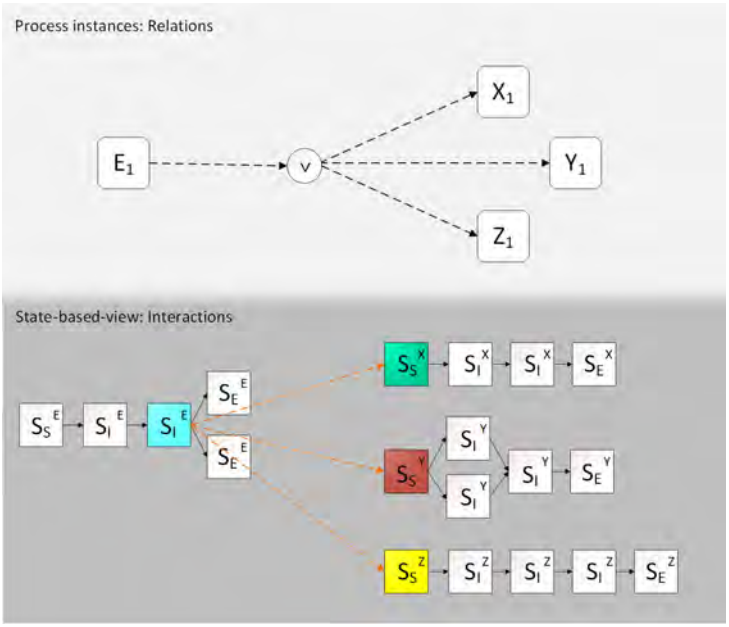
Prerequisite

P_{Source}= 1, $P_{Target}\geq$ 2

Description 1:1

The execution of one to multiple single process instances X_1, Y_1 and Z_1 depends on the execution status of one single process instance E_1. At least one of the single process instances X_1, Y_1 and Z_1 is executed. In the case where multiple single process instances are executed, execution may be concurrent.

Structure 1:1



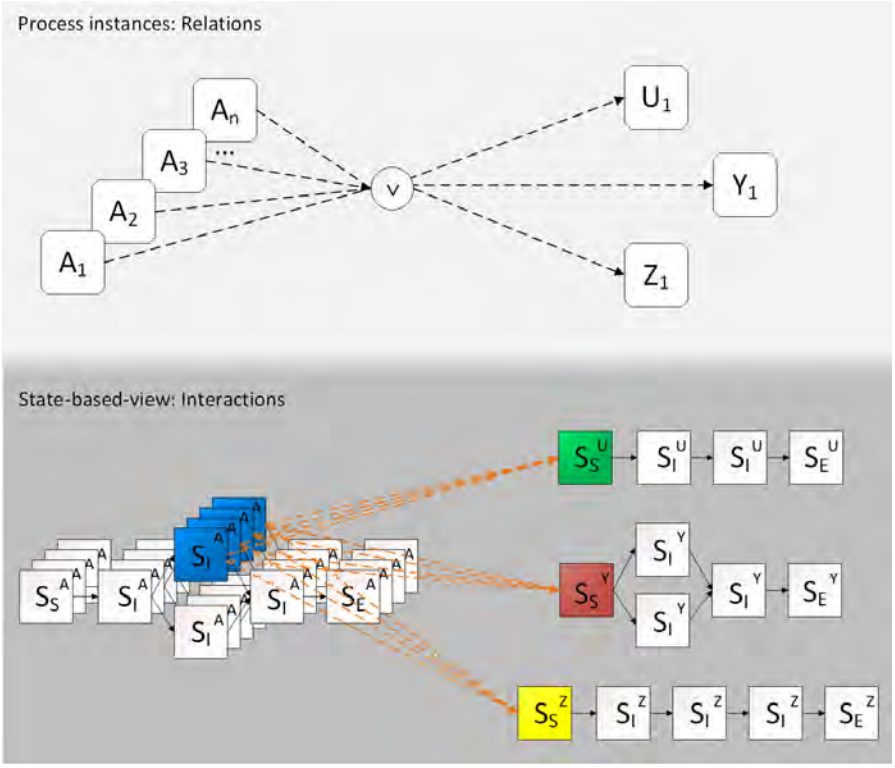
Example 1:1

A staff member must undergo a routinely health check. Based on the symptoms exhibited by the patient during the doctor's visit, one to several different lab tests may be conducted cf. [53].

Description m:1

The execution of one to multiple single process instances U_1, $Y_1$and Z_1 depends on the execution status of a set of process instances A_n. At least one of the single process instances U_1, $Y_1$and Z_1 is executed. In the case where multiple single process instances are executed, execution may be concurrent.

Structure m:1



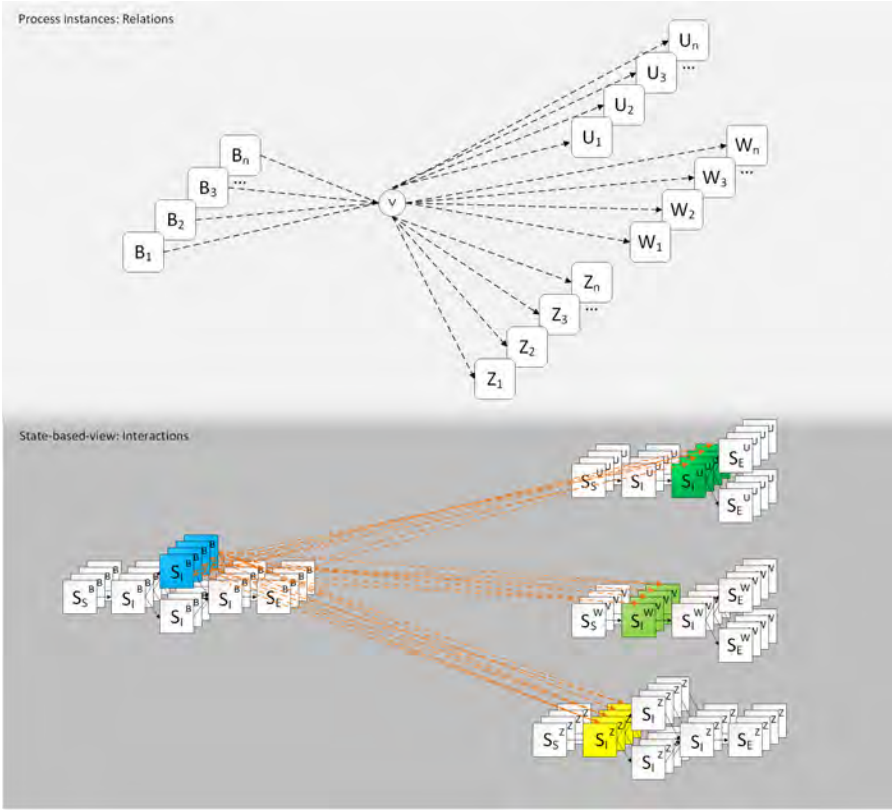
Example m:1

A staff member is submitted to an employee assessment. Depending on the results, the following measures may be taken: specialised training, transfer to another area and / or promotion.

Description m:n

The execution of one to multiple sets of process instances U_n, W_n and Z_n depends on the execution status of a set of process instances B_n. At least one of the sets of process instances U_n, W_n and Z_n is executed. In the case where multiple sets of process instances are executed, execution may be concurrent.

Structure m:n



Example m:n

Random samples are taken from a finished product. Based on the type of the defects, different spare parts need to be reordered.

Remark

Interaction indicator: the circle with a $\vee$ is used as visual aid to indicate that one source process type may interact with one to multiple target process types.

3.1.5. PCP 5 Synchronisation

Essentials

PCP 5 describes concurrent process interactions. Several source process types are related to one common target process type. The execution of the target process type may only proceed if all source process types have reached a specific execution status. All source process types may be executed concurrently.

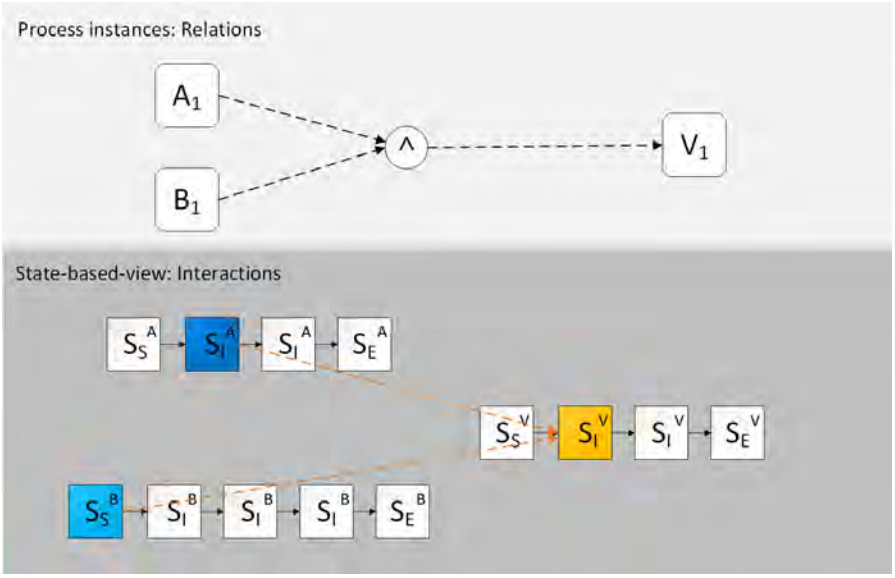
Prerequisite

$P_{Source} \geq 2$, $P_{Target} = 1$

Description 1:1

The execution of a single process instance V_1 depends on the execution status of two or more single process instances A_1 and B_1, which must be all executed. The single process instances A_1 and B_1 may be executed concurrently.

Structure 1:1



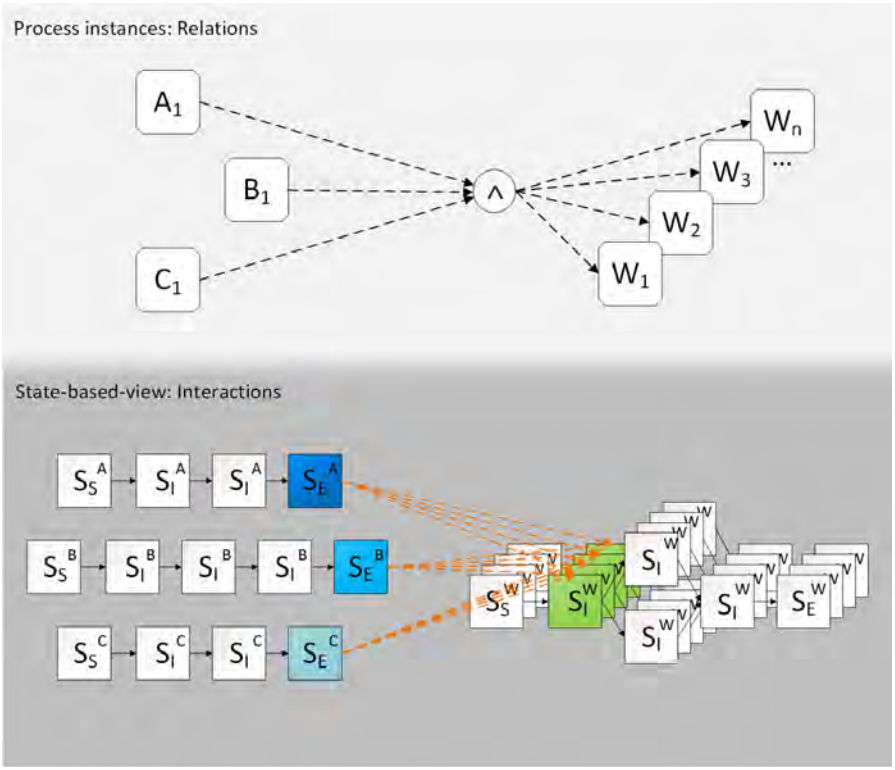
Example 1:1

A register customer places an order. Once the goods issue has been posted and the shipping process has been started, an invoice message is sent to the customer.

Description 1:n

The execution of a set of process instances W_n depends on the execution status of two or more single process instances A_1, B_1 and C_1, which must be all executed. The single process instances A_1, B_1 and C_1 may be executed concurrently.

Structure 1:n



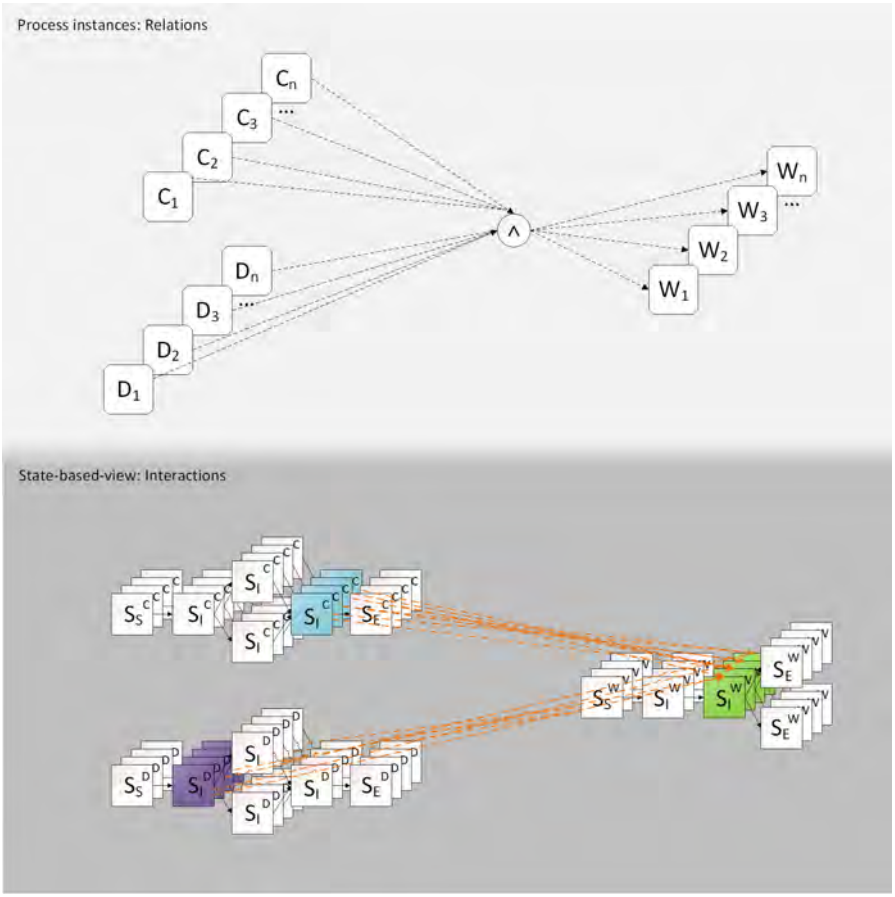
Example 1:n

In a build-to-order process of a computer manufacturer, customers order products that are custom-built. The manufacturer procures the required components from various suppliers. To reduce costs, the manufacturer bundles components of multiple customer orders in different joint purchase orders. The construction process of multiple customer orders can be started after all joint purchase orders have arrived [28].

Description m:n

The execution of a set of process instances W_n depends on the execution status of two or more sets of process instances C_n and D_n, which must be all executed. The sets of process instances C_n and D_n may be executed concurrently.

Structure m:n



Example m:n

The construction of a building is assigned to a company. After material-orders are placed and labourers are booked, the construction company can establish the detailed milestones for constructing the building.

Remark

Interaction indicator: the circle with a $\wedge$ is used as visual aid to indicate that all source process types may interact with a common target process type.

3.1.6. PCP 6 Selective Synchronisation

Essentials

PCP 6 describes a special case of concurrent process interactions. Several source process types are related to one common target process type. The execution of the target process type may proceed once a subset of the set of source process types have reached a specific execution status. Source process types are non- mutually exclusive. At least one source process type is executed. Subsets of the set of source process types may be executed concurrently.

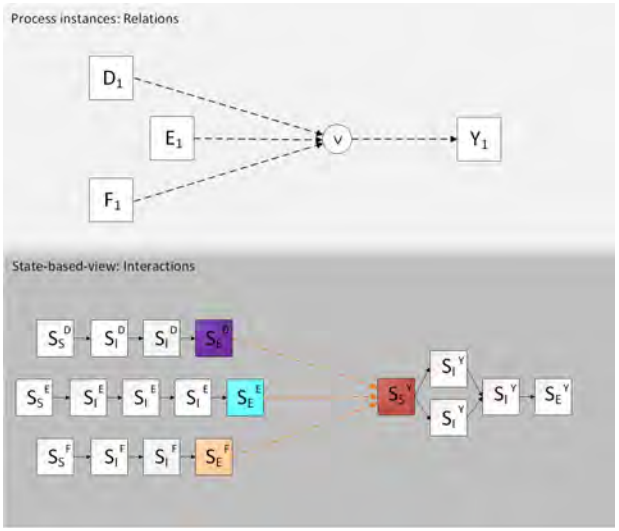
Prerequisite

$P_{Source} \geq 2$, $P_{Target} = 1$

Description 1:1

The execution of a single process instance Y_1 depends on the execution status of one to multiple single process instances D_1, E_1 and F_1. The single process instances D_1, E_1 and F_1 are non-mutually exclusive. At least one of the single process instances D_1, E_1 and F_1 is executed. In the case where multiple single process instances are executed, execution may be concurrent.

Structure 1:1



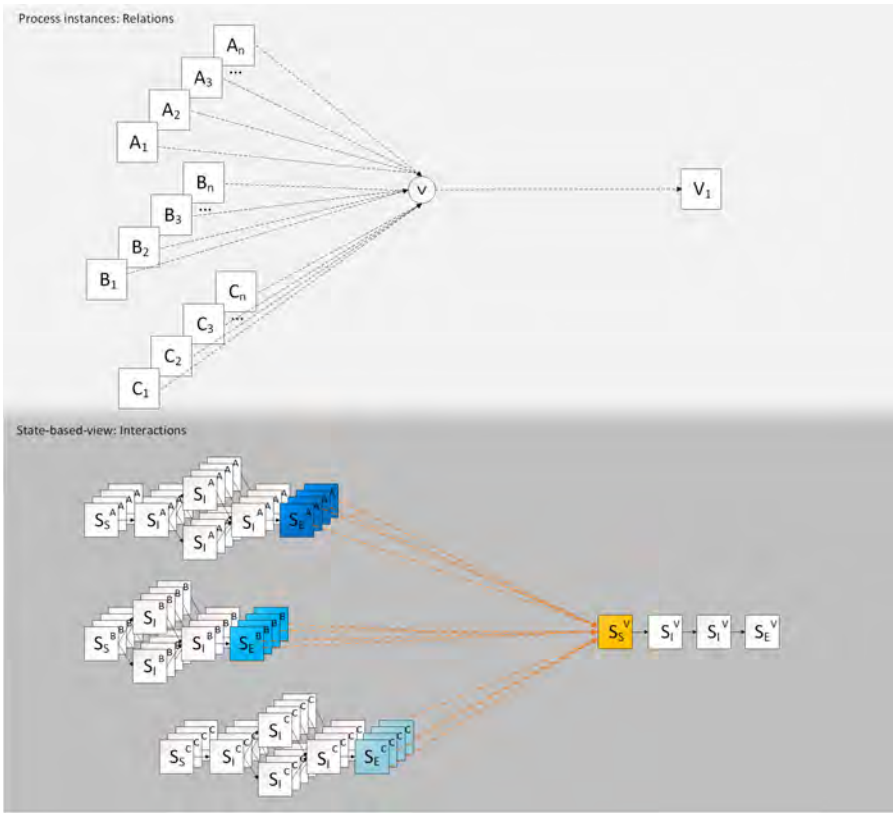
Example 1:1

A patient goes to a doctor for examination. Several kinds of tests are performed to check the presence of a suspected disease. The test results may be available at different times. Once a subset of tests has produced a positive result, a reliable diagnosis can be made and the patient may advance to treatment cf. [49].

Description m:1

The execution of a single process instance V_1 depends on the execution status of one to multiple sets of process instances A_n, B_n and C_n. The sets of process instances A_n, B_n and C_n are non-mutually exclusive. At least one of the sets of process instances A_n, B_n and C_n is executed. In the case where multiple sets of process instances are executed, execution may be concurrent.

Structure m:1



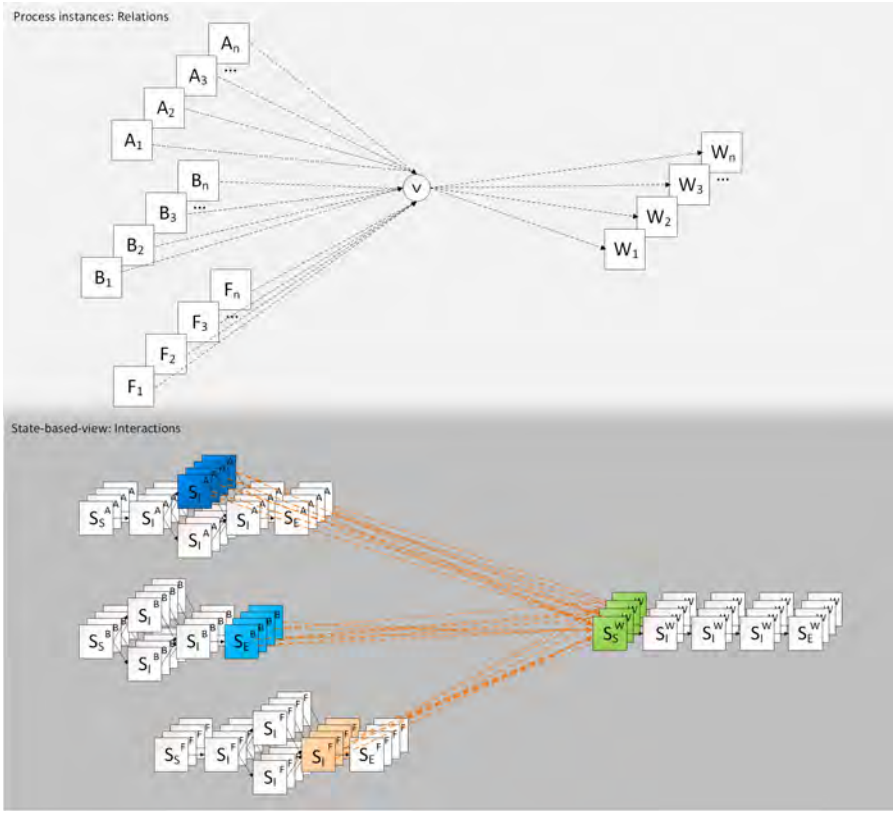
Example m:1

A travel agency offers the booking of different services as part of a comprehensive travel packaging. Once the booking processes for the hotel accomodations, car rentals and / or tickets to key attractions have been finalised, the payment process can be initialised.

Description m:n

The execution of a set of process instances W_n depends on the execution status of one to multiple sets of process instances A_n, B_n and F_n. The sets of process instances A_n, B_n and F_n are non-mutually exclusive. At least one of the sets of process instances A_n, B_n and F_n is executed. In the case where multiple sets of process instances are executed, execution may be concurrent.

Structure m:n



Example m:n

In the frame of a project on rural development in an African country, seeds are distributed to local farmers in different areas of the country. For ensuring food security, three to five different types of seeds shall be sown in each area. Several orders for each type of seed are placed. The sowing process in the different areas can start once the first type of seeds is handed over to the farmers.

Remark

Interaction indicator: the circle with a $\vee$ is used as visual aid to indicate that a subset of the set of source process types may interact with a common target process type.

3.1.7. PCP 7 Reassignment

Essentials

PCP 7 describes the dynamic change of relations among process instances during execution. A source process instance which is related to a certain target process instance is rerouted to a different target process instance. As a result, a relationship emerges which did not exist before.

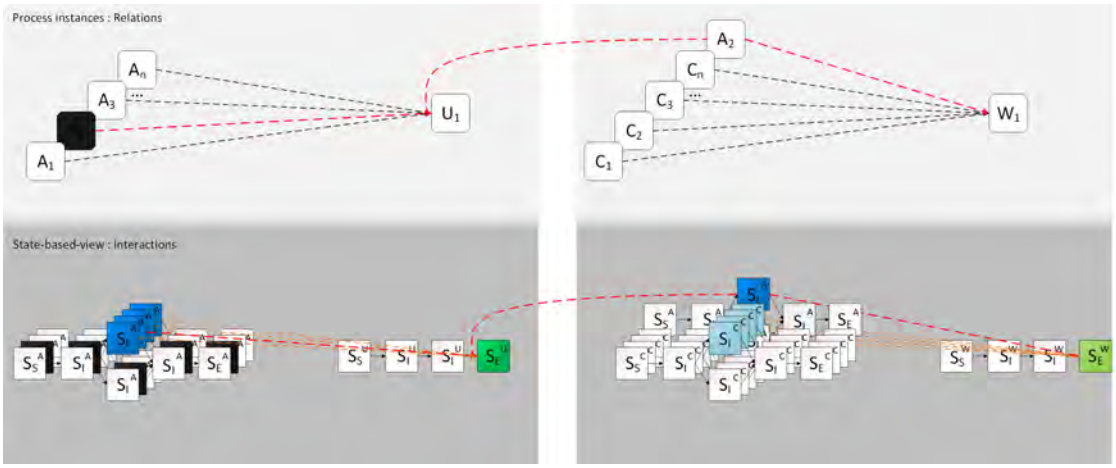
Prerequisite

$P_{Source} \geq 1$, $P_{Target} \geq 1$

Description 1:1

The execution of one single process instance U_1 depends on the execution status of a set of process instances A_n. The execution of another single process instance W_1 depends on the execution status of a set of process instances C_n. Of the set of process instances A_n at least one instance A_2 may be rerouted to a different process instance W_1 during execution. As a result, A_2 is now related to the process instance W_1 and added to the set of process instances C_n such that $C_n \ni A_2$.

Structure



Examples

- A company has published several job offers. During the review process of applications for a specific job offer, it turns out that the application of a certain candidate may be better suited for a different job offer which is also vacant. Therefore, the application is rerouted to this job offer for reviewing cf. [49].

- A potential tenant commissions a real state agent with finding an appartment for him. The real state agent has several applications for a specific property. During the selection process the potential tenant may be offered a different housing property than the originally applied for.
- In the process of selecting papers for a conference each paper is evaluated by various reviewers. A reviewer may declare that she might not meet the deadline for reviewing one or several papers assigned to her. As a result, these papers are assigned to a different reviewer for evaluation [54].

3.2. Acquired Insights during the Process of Pattern Discovery

During the compilation of the PCP catalogue, the relationships between the discovered seven patterns were analysed. It has been established that based on their similarities the patterns can be divided into four groups.

The first pattern group consists of one pattern, *PCP 1* which addresses the basic case of process interaction involving only two process types.

The second group consists of three patterns. *PCP 2, PCP 3* and *PCP 4* follow the same scheme, where the execution of several target process types depends on the execution status of one source process type, namely:

- In the case of *PCP 2* the entire set of target process types is executed.
- In the case of *PCP 3* at most one target process type of the entire set is executed.
- In the case of *PCP 4* one to multiple target process types are executed.

The third pattern group consist of two patterns. *PCP 5* and *PCP 6* follow a common scheme, where the execution of one target process type depends on the execution status of several source process types, in particular:

- In the case of *PCP 5* the entire set of source process types is executed.
- In the case of *PCP 6* one to multiple source process types are executed.

The fourth group consists of one pattern, *PCP 7*, which compared to the other groups seems to be out of the ordinary. *PCP 7* has been added to the catalogue after being observed in several scenarios, but its clustering not being suitable in any of the other pattern groups. While the other six PCPs specify one scenario with related source and process types, *PCP 7* assumes more than one scenario of related source and process types. Process instances that are part of one relation in one scenario are rerouted and attached to another relation in another scenario.

Concerning the second and third group of patterns, those may be seen as converse cases. While *PCP 2* and *PCP 5,* and *PCP 4* and *PCP 6* represent appropriate counterparts, *PCP 3* remains without a counterpart. This results from the fact that, a pattern where the execution of a common target process type depends on the execution status of at most one source process type out of the entire set seems non-existent. Such a pattern implies that the execution of the target process type requires one input but not the other. To be more precise, the execution of the target process type may only proceed if at most one source process type out of the entire set has reached a specific execution status, such that the remaining ones are prevented from being executed. However, based on how PCPs are defined the described pattern seems non-existent in business scenarios involving different process types related to one another. It seems highly unlikely that the execution of one target process type may depend on at most one out of several

source process types, such that only one source process type can be executed. For such a scenario it would be more consistent to apply *PCP 1* as the execution of the target process type depends only on one source process type, such that the existence of further source process types is irrelevant.

Each one of the proposed seven PCPs intends to address a recurring type of interaction among related processes encountered in business scenarios. The PCP catalogue compiles elementary interactions; for more complex interactions to be modelled a number of PCPs may need to be combined.

4

Evaluating the Support of Process Coordination Patterns

In this chapter, two modelling approaches, of which one is based on the activity-centric paradigm and the other on the data-centric paradigm, are assessed for the support of the current PCP catalogue. Two scenarios exhibiting multiple process interactions, in particular one-to-many and many-to-many relationships, are exemplarily used for the evaluation. The degree of pattern support for each modelling approach is determined based on a 3-value scale. The obtained results are discussed, investigating the encountered problems and corresponding reasons.

4.1. Evaluation Methodology

One of the values of PCPs lies in their independence from specific BPM paradigms. Therefore, two approaches, based on different paradigms, are assessed for their PCP support in this section. As mentioned in Chapter 1, the main paradigms implemented in PrMS are the widely used activity-centric paradigm, the case-handling paradigm and the more recent data-centric paradigm.

Prior to the assessment of modelling approaches, a framework determining relevant criteria for their systematic selection was established. It is emphasised that, for the evaluation of modelling approaches on their suitability for supporting PCPs, a strong tool support for modelling is indispensable. The framework applied for the selection of the modelling approaches is based on the following criteria:

- ***C1-Behaviour specification***: Capability of the approach to model the behaviour of a process from its creation to its end.
- ***C2-Interaction specification:*** Refers to the presence of an interaction concept.
- ***C3-Support for model verification:*** Capability of the approach for verifying a model in respect to a specified set of correctness criteria (e.g. absence of deadlocks and livelocks).
- ***C4-Tool support for modelling:*** Refers to the availability of a (GUI-based) tool that supports the design of process models.

Based on these criteria for assessing PCP support, case-handling approaches are irrelevant for PCPs for fundamental reasons. Case-handling approaches are based on the concept of an isolated case containing all activities and data objects that are part of it. Consequently, cases cannot have interactions, so that criterion *C2-Interaction specification* is not supported [50]. Therefore, from the considered approaches for evaluation, one is based on the activity-centric paradigm and the other on the data-centric paradigm.

The described criteria, among others, have been assessed in detail in [50] in a systematic literature review (SLR). Based on the conducted SLR, in [50] a framework for the systematic evaluation and comparison of data-centric approaches was developed. In the SLR 17 process modelling approaches identified from 38 primary studies were thoroughly analysed. The SLR provides insights of the capabilities of several data-centric approaches; thereby a special focus was put on the tooling and software support of the approaches. The data-centric approaches analysed in the SLR, which exhibit a mature tool implementation, were taken into account for the selection of the data-centric modelling approach evaluated in this thesis.

The evaluation of the criteria proposed in [50] is based on a 3-value scale consisting of the values: *fully supported, partially supported* and *not supported*. Concerning criterion *C1*, an approach fully supports behaviour specification if it enables the formal specification of a behaviour model at design-time. In the case of a partially formal or informal behaviour model specification, the approach exhibits partial support. Regarding criterion *C2*, the approach fully supports interaction specification if the interaction concept is completely formalised. If the interaction concept is partially formalised or informal specifications exist, this criterion is partially supported. Referring to criterion *C3*, full support is given if it can be evaluated whether all aspects of a process model are compliant with formally specified correctness criteria. Partial support is assigned to the approach in the case that an aspect of the process model lacks formalised criteria or the correctness criteria are only stated informally, such that formal verification is not possible. Relating to criterion *C4*, full support for modelling indicates the presence of a tool that allows specifying all aspects of a process model. If the tool supports at least one but not all modelling aspects, the support of the approach is considered as partial. If no tool exists, criterion *C4* is considered as not supported.

Based on the results of the SLR, the modelling notation of the PHILharmonicFlows framework has been selected for the evaluation of PCP support in this thesis. According to the SLR, this approach fully supports the four criteria stated above. In the following, a brief overview of the PHILharmonicFlows approach is provided.

PHILharmonicFlows [25] is classified as a data-centric approach for BPM. The PHILharmonicFlows approach is based on the object-aware concept, where business processes are specified in terms of interdependent objects with lifecycles that interact with one another, based on established relations between them. Objects are the main building blocks of the PHILharmonicFlows approach. They represent business objects such as a customer order or a job application. Objects hold attributes and have a lifecycle

process, in which the object's behaviour is described. Objects alone do not constitute a business process, but it is only through interactions among objects that a business process emerges. At certain points in time for the business process to advance, the progress of an object may depend on the execution status of another object. On this account, the behaviour of the involved objects needs to be coordinated, i.e. interactions between objects are necessary.

In this context, PHILharmonicFlows differentiates three model types: the data model [48], the lifecycle processes [46] and the coordination processes [47]. In the data model, objects, their attributes and relations to other objects are captured. Relations connecting objects with one another are similar to ER modelling, but additionally indicate process dependencies. For example an object such as *product* may be connected to another object such as *customer order* through a 1:n relation. At design-time, this relation indicates that many products may be related to one customer order. At run-time, based on the defined 1:n relation, interactions between the involved objects may take place by coordinating their behaviours. Furthermore, relations between objects may be restricted by assigning cardinality constraints. A lifecycle process consists of different states that are connected by state transitions. Objects progress from one state to another, based on available data, i.e. a state becomes executable once required attribute values are present. However, at times the execution state of an object's lifecycle process may depend on the execution state of another object's lifecycle process. Coordination processes capture these object interdependencies in the form of *semantic relationships*, which represent basic dependency constraints among processes in one-to-many and many-to-many relationships. At run-time, coordination processes enforce semantic relationships based on the execution status of the objects between which semantic relationships are defined. In other words, an object's process execution may progress or halt based on whether conditions captured by semantic relationships are currently satisfied or not.

Hereafter it is referenced how according to the SLR performed in [50], PHILharmonicFlows supports the four criteria *C1, C2, C3* and *C4* stated above. The formal specification of lifecycle processes, describing objects' behaviour, awards full support for criterion *C1*. Coordination processes capture the overall business processing logic, which is based on the interactions of the different objects, involved at specific points in time. The presence of a formal specification for interactions makes criterion *C2* fully supported. In the PHILharmonicFlows framework the data model, the lifecycle processes and the coordination processes are formally specified. Formal correctness criteria can be defined, thus enabling a complete verification of all specified process models of the approach. Thereby, this approach is fully compliant with criterion *C3*. The tooling for the PHILharmonicFlows framework comprises a modelling tool for the creation and verification of process models. Therefore, full support is granted to criterion *C4.*

Concerning the activity-centric approach selected for evaluation in this thesis, the Business Process Model and Notation (BPMN) as the de-facto standard for business process modelling has been selected. In the following, the main characteristics of BPMN are described, pointing out how BPMN supports the four criteria *C1, C2, C3* and *C4*. BPMN was

not assessed in [50], however for the evaluation of the criteria the technical specification of BPMN [35] was used.

BPMN is a graphical notation based on the imperative activity-centric paradigm, wherein only allowed activity flows are captured in the process model. The created process models describe one primary way of performing business processes which are predictable and may have relatively few and well-scoped variations [19, 61].

In models created with BPMN, every process is contained in a *pool* describing the sequence in which activities take place within a single organisation. A single pool contains *private business processes*, i.e. these processes are internal to a specific organisation. The *sequence flow* contained within a single pool cannot cross the boundaries of the pool. The formal specification of pools, the therein contained activities and sequence flow describing the behaviour of a process award full support for criterion *C1*. In order to model cross-organisational business processes, also known as collaborations, several pools are used. Each partner's process is modelled in a separate pool such that several partner's private business processes interact with one another to reach a common business goal. Consequently, interactions only exist between separate private business processes. Interactions are modelled and coordinated with *message flows* crossing pool boundaries, connecting single processes with one another. In addition, *data artifacts* representing data objects or data stores can be used to show the flow of information within and across pool boundaries [35]. Since a formal specification for interactions exists, criterion *C2* is fully supported. In the BPMN technical specification the basic execution semantics of BPMN elements such as events, gateways, data artifacts etc. is described informally (textually). The data type definition language for execution semantics is canonically XML-schema. However, the formal verification of the sequence flow and data flow is outside the scope of the BPMN technical specification. As a result, diverse verification approaches for detecting sequence flow and data flow anomalies have emerged. Examples for verification techniques are mapping BPMN models to Petri nets [2, 12, 45] or to YAWL [9], an extension of Petri nets. Since the BPMN standard does not provide a generally valid verification framework, criterion *C3* is partially supported. For BPMN several management-suites with an integrated modelling tool exist, thereby criterion *C4* scores full support.

PHILharmonicFlows as a representative of the data-centric paradigm and BPMN as a representative of the activity-centric paradigm were assessed for PCP support. For this purpose, two scenarios, the paper selection process of a conference and a built-to-order process were modelled with each of the selected approaches. These scenarios were chosen because in total six out of the seven PCPs compiled in the current catalogue are present in the scenarios. Each scenario is described in detail and the PCPs found therein are listed. The suitability of PHILharmonicFlows and BPMN for modelling complex process interactions was evaluated by investigating which of the PCPs found in the scenarios are directly supported.

As shown in Table 4.1 the degree of pattern support is determined based on how the criteria *relationship type*, *cardinality restriction* and *threshold value* ω are supported in

terms of modelling. A 3-value scale consisting of the values: *direct support*, *partial support* and *no support* is hereby assigned.

Tabelle 4.1.: Evaluation Criteria for Pattern Support

	Relationship type			**Cardinality restriction**	**Threshold value ω**
	$1:1$	$1:n/m:1$	$m:n$	$m_{\text{lower}}, m_{\text{upper}}, n_{\text{lower}}, n_{\text{upper}}$	Assignment at design-/run-time
Direct support	**+**	**+**	**+**	**+**	**+**
Partial support	**+**	**+**	**+**	**+/-**	**+/-**
No support	**-**	**-**	**-**	**-**	**-**

A pattern is directly supported if it can be properly represented with the modelling approach. This requires that, the type of relationship between interacting instances of different processes can be distinguished. Furthermore, the modelling approach allows to set cardinality restrictions for specifying the number of source process instances that can be related to a number of target process instances at run-time. In addition, the threshold value ω determining the condition for process interactions can be explicitly assigned through formal constructs of the modelling approach at design- and / or run-time.

Partial support is awarded if the pattern can be indirectly represented with the modelling approach. This means that, the type of relationship between interacting instances of different processes can be distinguished. However, the modelling approach may not support cardinality restrictions, so that the number of source and target process instances related to one another is undefined. Furthermore, the assignment of the threshold value ω may not be explicitly supported through formal constructs. It is conceivable that cardinality restrictions and threshold value assignment may be represented with a workaround such as a textual annotation. A pattern is not supported if it cannot be modelled.

4.2. Scenario 1: Paper Selection Process of a Conference

Academic conferences offer researchers the opportunity to publish and discuss their work. Conferences must ensure that published papers fit the topic of the conference and meet certain quality criteria. For this purpose, conferences implement a rigorous selection process for papers, based on “peer reviews”. The goal of the paper selection process is selecting high-quality papers out of a set of submitted papers. The number

of papers to accept is either fixed beforehand, or a desired percentage ("acceptance rate") of all submitted papers is defined. Academic conferences usually consist of an organising committee with one or more general chairs, a program board (PB) and a program committee (PC). PB members are responsible for the management of the reviews, while PC members are responsible for performing the peer reviews.

4.2.1. Process Description

In the following, a generic selection process for an academic conference, adapted from [7, 54], is described:

Initial conference organisation: At first, a set of people is appointed by the general chairs to participate as members of the PB and PC. The appointed individuals may accept or decline to join the program board and program committee; for those that decline replacements need to be found.

Call for papers: Once a certain number of PB and PC members is staffed, a call for papers is discussed and agreed upon. The call is then published, announcing the upcoming conference and asking for submissions. The call for papers states the topics of the conference as keywords and explains the paper selection process. It includes the key dates of the conference such as the paper submission deadline, the acceptance notification date, the camera-ready paper deadline and the actual conference dates. Furthermore, the specific layout and maximum length of the paper is defined in the call for papers.

Paper submission: Authors prepare and submit their papers according to the call for papers. This is required to happen on or before the submission deadline.

Pre-Check and Bidding: After the submission period has ended, PB members check whether all submitted papers meet the formal criteria (paper layout, page limit etc.). Papers which do not comply with the requisites are rejected and do not make it to the bidding phase. In the bidding phase, PB and PC members, choose their preferences for papers they want to review, based on their expertise and interests. Furthermore, they must declare possible conflicts of interest.

Paper distribution: At the end of the bidding phase, an algorithm allocates a subset of papers to each PC member. The decision how to distribute the papers among PC members takes into account the whole set of available papers. The allocation is best effort; i.e. PC members are not guaranteed to only receive papers they placed a bid for, although they will never receive papers from a conflict of interest author. Also, each PB member is responsible for at least one submitted paper, which requires performing a meta-review based on the submitted reviews.

Review: PC members should review the assigned papers on time. They may assign papers to sub-reviewers at their own discretion. Usually, it is ensured that each paper receives at least three reviews. If reviewers are unable to deliver the assigned reviews,

additional reviewers may be appointed by PB members in order to ensure a fair evaluation. Once the reviews are submitted, PB members start preparing the meta-reviews. In case that not all three reviews per paper are received, the meta-review is made based on the submitted ones. Meta-reviews recommend whether to accept or reject a paper. This process should take place weeks prior to the notification key date.

Discussion and paper selection: In this phase a final assessment of the submitted papers is made based on submitted meta-reviews. The papers are compared and ranked by PB members, selecting those of acceptable quality. Another selection criterion is the minimum and maximum number of papers to be presented. In the case where several papers address the same topic from different or opposing perspectives, this needs to be considered. Once the final set of accepted submissions has been determined, all other submissions are rejected. Finally, if necessary, the responsible PB member has to update the meta-review for making the decision of acceptance or rejection transparent to the authors.

Notification: Authors are notified by e-mail of either acceptance or rejection of their papers. In case of acceptance, authors must prepare the camera-ready version of their paper ensuring correct formatting. Furthermore, reviewer's comments must be taken into account. If authors choose to ignore the reviewer's comments, it may lead to the rejection of the paper. Authors must sent their camera-ready version on or before the corresponding deadline. Otherwise, the submission will not be part of the proceeding's publication.

Publication: Timely submitted final versions are assembled and sent to the publisher for preparing the conference proceedings.

4.2.2. Patterns Overview

In the described process of paper selection of a conference the following fundamental process types with their corresponding states were abstracted:

- Conference (*C*): Preparation, Publication, 1. Closure, 2. Closure, 3. Closure, Proceedings Assemble.
- Paper (*P*): Creation, Submission, Initial Check, Selection, Camera-Ready Preparation, Camera-Ready Submission, Camera-Ready Check, Collection for Publication.
- Bid (*B*): Creation, Preparation, Placement.
- Review (*R*): Assignment, Paper Assessment, Recommendation, Submission.
- Meta-review (*MR*): Preparation, Selection proposal, Submission.

The overall business process exhibits interactions between the involved process types such that two PCPs can be found. *PCP 1* appears seven times, while *PCP 3* is found once; altogether eight PCPs are comprised in the scenario. In the following these PCPs are described.

PCP No.: #1

PCP: PCP 1

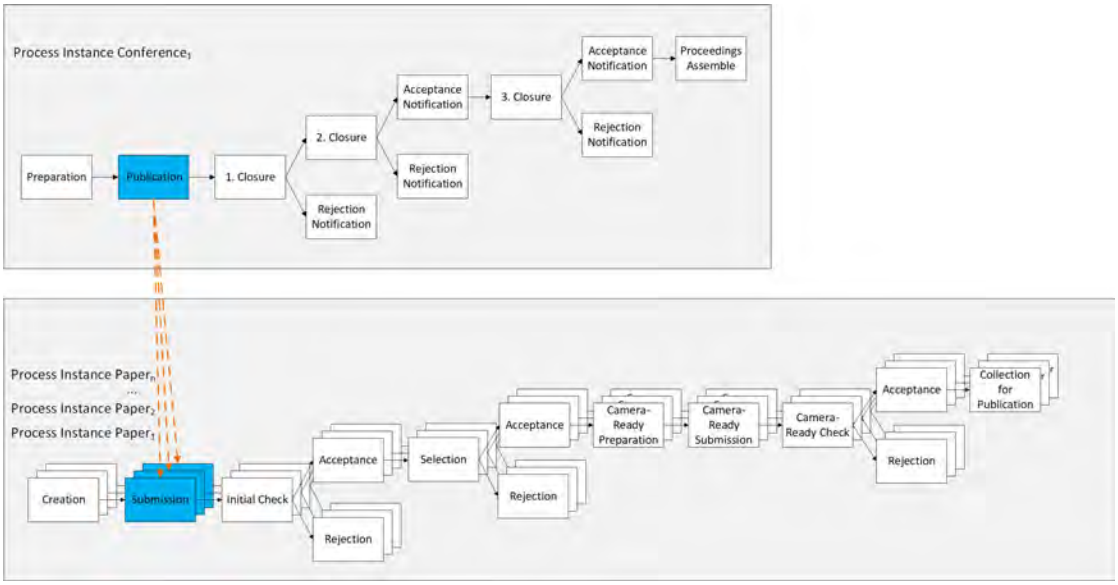
Process Types: Conference, Paper

Description: Once the conference is published, several papers may be submitted at different points in time.

Relation Type: $1:n$

Cardinality Restriction: $m^{C}{}_{\text{lower}} = m^{C}{}_{\text{upper}} = 1; n^{P}{}_{\text{lower}} = n^{P}{}_{\text{upper}} = n$

Threshold value: $\omega^{C} = 1$



PCP No.: # 2

PCP: PCP 3

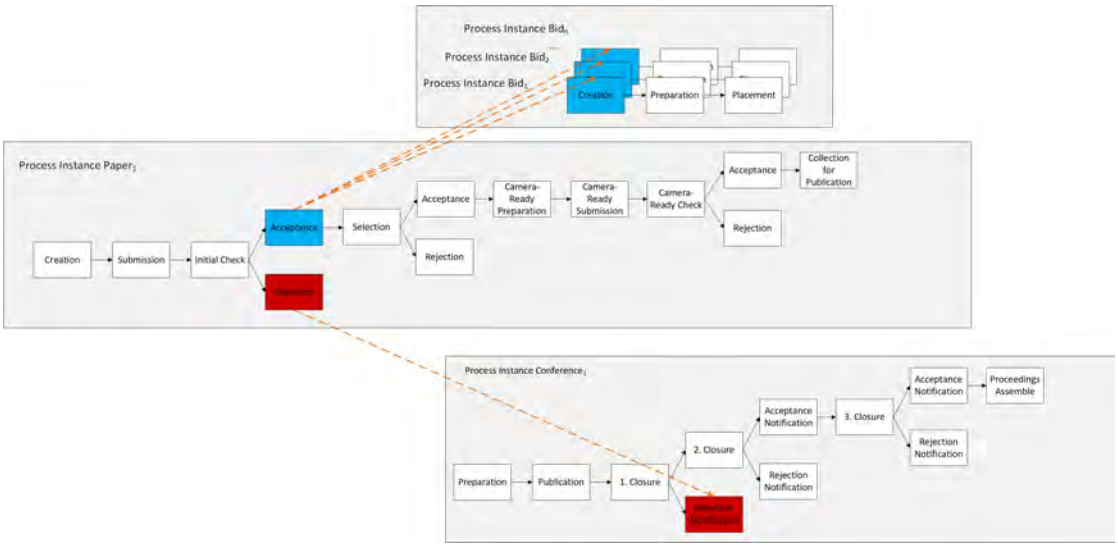
Process Types: Paper, Bid, Conference

Description: Based on the result of the initial paper check, either bids are placed for reviewing the paper or an initial rejection notification is sent.

Relation Type: $1:n \parallel 1:1$

Cardinality Restriction: $m^{P}{}_{\text{lower}} = m^{P}{}_{\text{upper}} = 1; n^{B}{}_{\text{lower}} = n^{B}{}_{\text{upper}} = n,$ $n^{C}{}_{\text{lower}} = n^{C}{}_{\text{upper}} = 1,$

Threshold value: $\omega^{P} = 1$



PCP No.: # 3

PCP: PCP 1

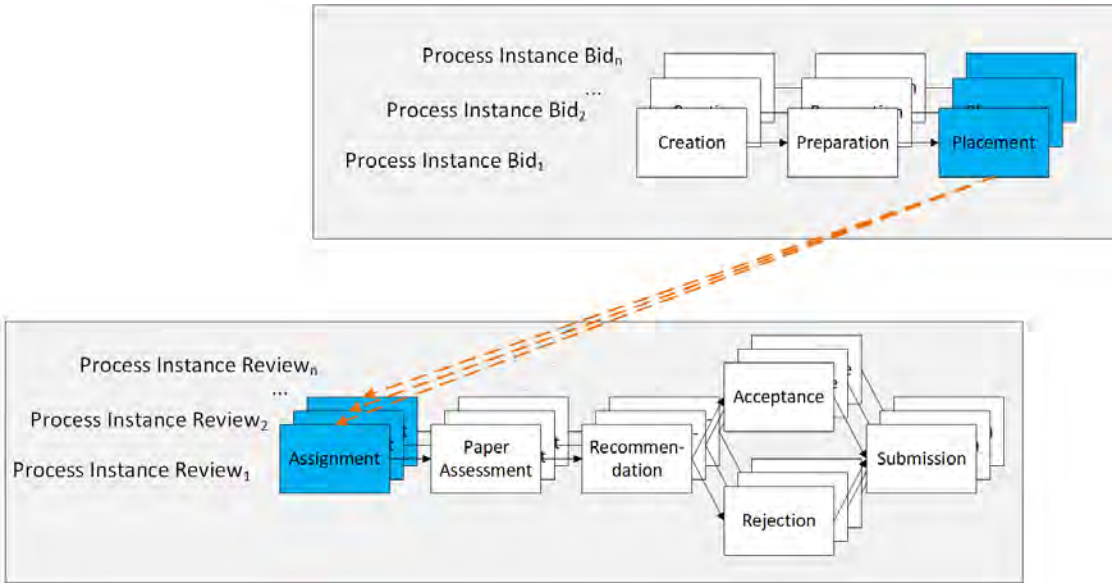
Process Types: Bid, Review

Description: Once bids for the whole set of available papers are placed, reviews may be assigned.

Relation Type: $m : n$

Cardinality Restriction: ${m^B}_{\text{lower}} = {m^B}_{\text{upper}} = m$; ${n^R}_{\text{lower}} = {n^R}_{\text{upper}} = n$,

Threshold value: $\omega^B = m$



PCP No.: # 4

PCP: PCP 1

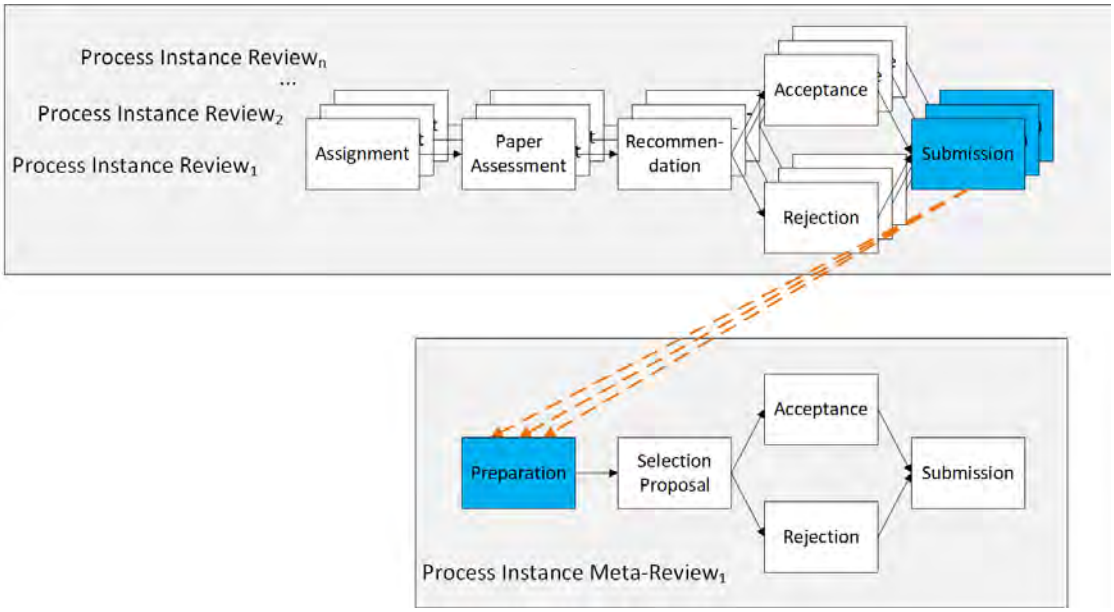
Process Types: Review, Meta-Review

Description: Once the reviews related to a paper are submitted, a meta-review may be prepared. At least, if possible, three reviews should be submitted for each paper before a meta-review can be prepared.

Relation Type: $m : 1$

Cardinality Restriction: $m^{R}_{\text{lower}} = 3, m^{R}_{\text{upper}} = 5; n^{MR}_{\text{lower}} = n^{MR}_{\text{upper}} = 1,$

Threshold value: $\omega^{R} \geq 3$



PCP No.: # 5

PCP: PCP 1

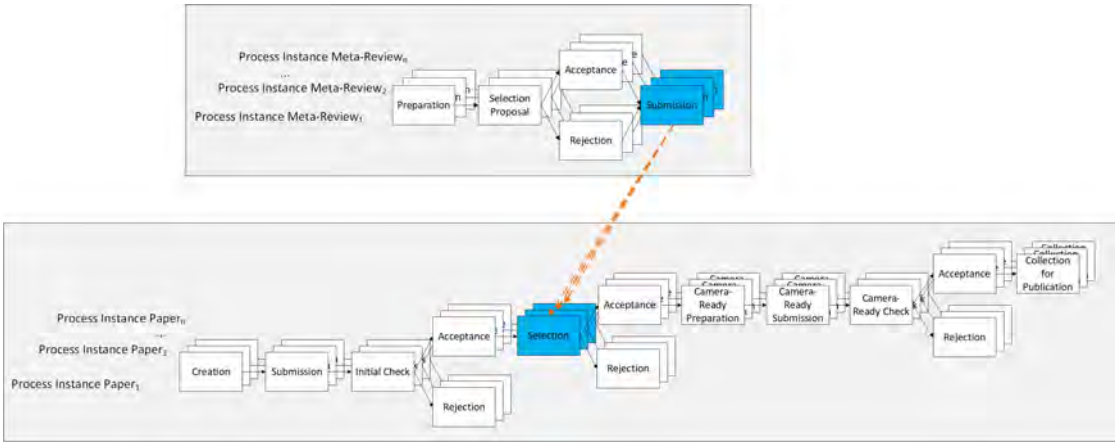
Process Types: Meta-Review, Paper

Description: Once all meta-reviews for the whole set of available papers are submitted, the final set of papers may be selected.

Relation Type: $m : n$

Cardinality Restriction: $m^{MR}{}_{\text{lower}} = m^{MR}{}_{\text{upper}} = m; n^{P}{}_{\text{lower}} = n^{P}{}_{\text{upper}} = n,$

Threshold value: $\omega^{MR} = m$



PCP No.: # 6

PCP: PCP 1

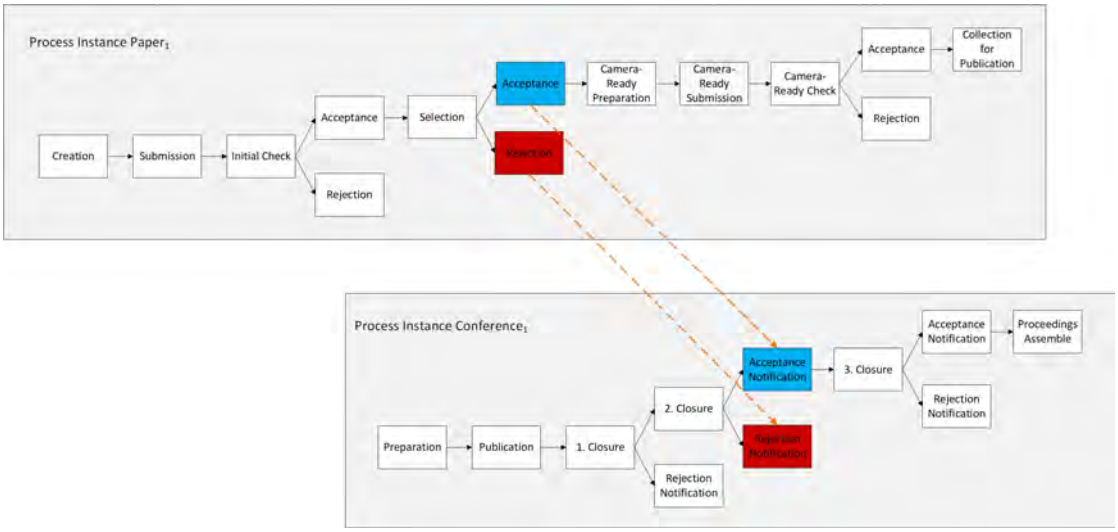
Process Types: Paper, Conference

Description: Once a paper has been assessed in the final selection process, a notification stating the decision may be sent. A paper is either selected or rejected.

Relation Type: $1:1$

Cardinality Restriction: $m^{P}{}_{\text{lower}} = m^{P}{}_{\text{upper}} = 1; n^{C}{}_{\text{lower}} = n^{C}{}_{\text{upper}} = 1,$

Threshold value: $\omega^{P} = 1$



PCP No.: # 7

PCP: PCP 1

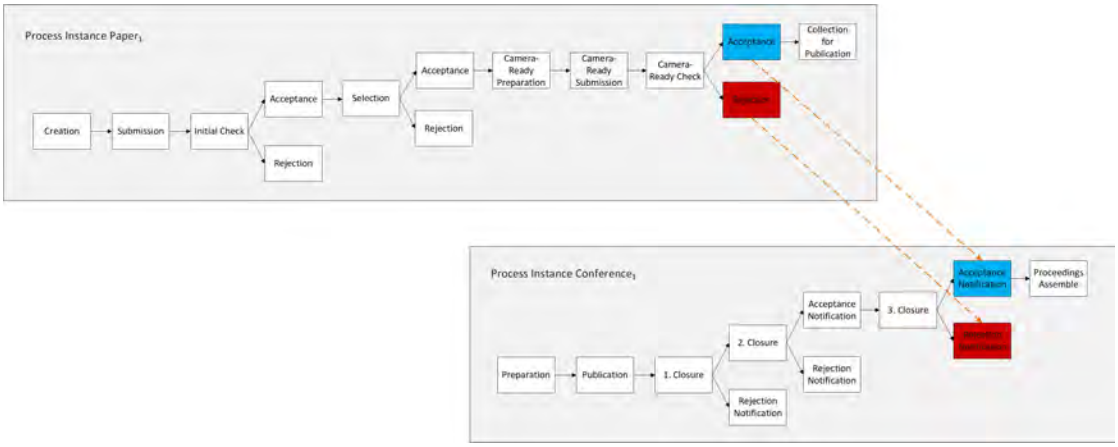
Process Types: Paper, Conference

Description: Once the camera-ready version of a paper has been assessed, a final rejection or acceptance notification may be sent.

Relation Type: $1:1$

Cardinality Restriction: $m^{P}{}_{\text{lower}} = m^{P}{}_{\text{upper}} = 1; n^{C}{}_{\text{lower}} = n^{C}{}_{\text{upper}} = 1,$

Threshold value: $\omega^{P} = 1$



PCP No.: # 8

PCP: PCP 1

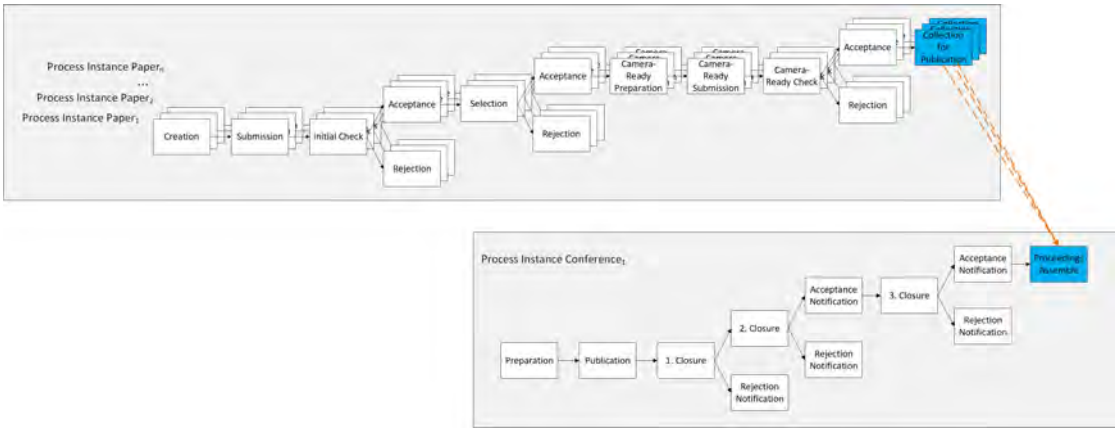
Process Types: Paper, Conference

Description: Once the final paper versions are collected for publication, the conference proceedings may be prepared.

Relation Type: $m:1$

Cardinality Restriction: ${m^P}_{\text{lower}} = {m^P}_{\text{upper}} = m$; ${n^C}_{\text{lower}} = {n^C}_{\text{upper}} = 1$,

Threshold value: $\omega^P = m$



4.3. Scenario 2: Built-to-Order Process

Built-to-Order (BTO) is a production approach where products are custom-made. BTO products usually exceed standard specifications, therefore products are not built until a confirmed customer's order is received. BTO production is typically found in highly customisable products such as computers and cars. For economic reasons, manufacturers of BTO products usually do not store specialised components in a warehouse but source them from suppliers according to customer orders.

4.3.1. Process Description

Exemplarily, a BTO process, adapted from [28, 36], is described with a custom-built PC offered by an online retailer:

Customer order registration: The process starts when the customer selects the PC configurator on the retailer's website. At first, the basic template of the PC system is chosen by the customer; i.e. whether it is a gamer-PC, mini-PC, a PC for professional image editing etc. The template system usually comes with a default configuration, but may be individually configured. Thereby it is distinguished between mandatory basic components and optional components. The customer chooses his or her desired components among the available options of the retailer for the following basic components: computer case, mainboard, processor, RAM, CPU cooler, graphics card, power supply unit, and hard drive/SSD. Optional components available for selection are, among others, the operating system, monitor, peripherals, additional hard drives, and high-performance cooling solutions, e.g. water cooling. If the customer is satisfied with the chosen configuration, the customer confirms the order and proceeds with the online payment. The online retailer then proceeds to source the PC components. For simplicity, the sourcing process starts once payment has been received.

Components sourcing: Components are not held in stock but need to be sourced from suppliers. The online retailer creates a number of purchase orders to be send to various suppliers to procure the required components. To reduce costs, components of multiple customer orders are bundled into joint purchase orders (JPOs). This means that, one JPO may contain components of multiple customer orders. Thus, one customer order depends on the components procured with multiple JPOs. A JPO is sent once a minimum order value for the corresponding component has been reached. Once a JPO is received by the corresponding supplier, it is checked against the stock. Based on stock availability suppliers can confirm or reject the orders. If a JPO is rejected by a supplier, then a new JPO is sent to another supplier for these components. If a JPO order is confirmed, the components are retrieved from the warehouse. The components are then shipped and an invoice is sent to the online retailer.

Customer order fulfilment: Once all JPOs are delivered, the components are assembled into the ordered custom computer. After assembly, every configured PC is subject to a

rigorous testing, ensuring functionality and stability. The product is then delivered to the customer and the customer order is archived.

4.3.2. Patterns Overview

In the described process of Built-to-Order custom-PC the following fundamental process types with their corresponding states were abstracted:

- Custom-PC (*PC*): Basic Template Selection, Basic Components Selection, Optional Components Selection, Confirmation, Online Payment, Assembly, Testing, Delivery
- Joint Purchase Order (*JPO*): Creation, Sent, Confirmation, Bill
- Basic Components (*BC*): Selection, Register, Order Placement, Retrieval from Warehouse, Shipment, Delivered, Installed
- Optional Components (*BC*): Selection, Register, Order Placement, Retrieval from Warehouse, Shipment, Delivered, Installed

In order to avoid any repetition, the process types of the individual basic and optional components have not been described for each one, for they have the same execution states. The overall business process exhibits interactions between the involved process types such that four PCPs can be found. *PCP 2* and *PCP 5* appear four times each, while *PCP 4* and *PCP 7* are each found once; altogether ten PCPs are comprised in the scenario. In the following those PCPs are described.

PCP No.: #1

PCP: PCP 2

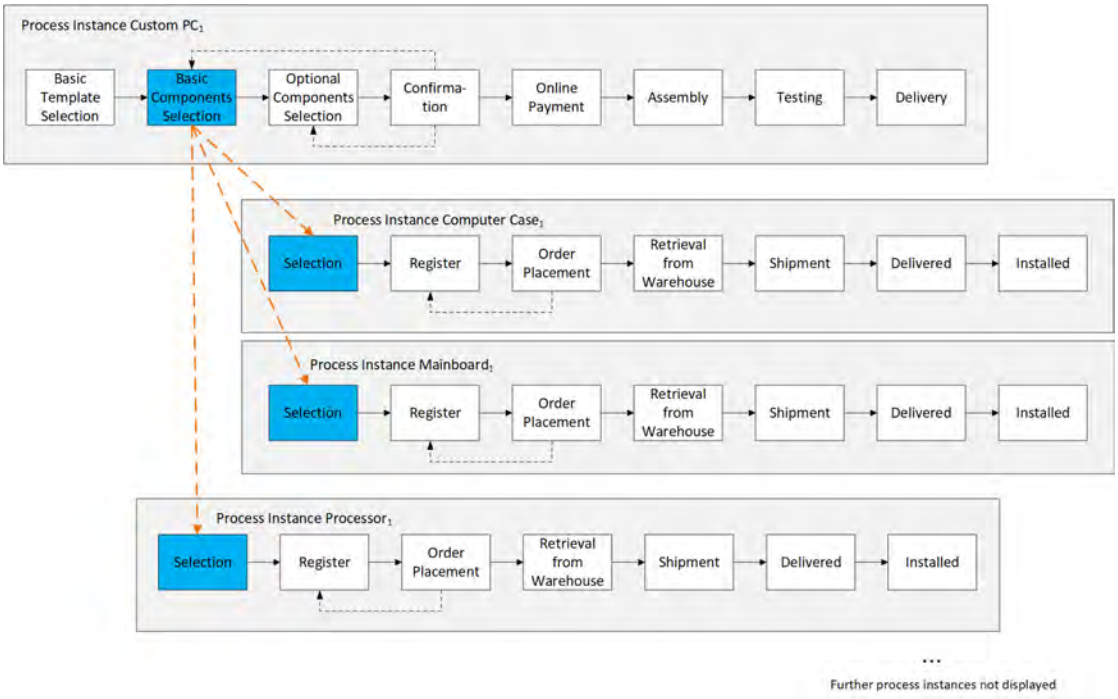
Process Types: Custom-PC, Basic Components

Description: Once the basic components of the custom-PC are configured in the PC configurator by the customer, these may be marked as selected.

Relation Type: $1:n$

Cardinality Restriction: $m^{PC}_{\text{lower}} = m^{PC}_{\text{upper}} = 1; n^{BC}_{\text{lower}} = n^{BC}_{\text{upper}} = n$

Threshold value: $\omega^{PC} = 1$



PCP No.: #2

PCP: PCP 4

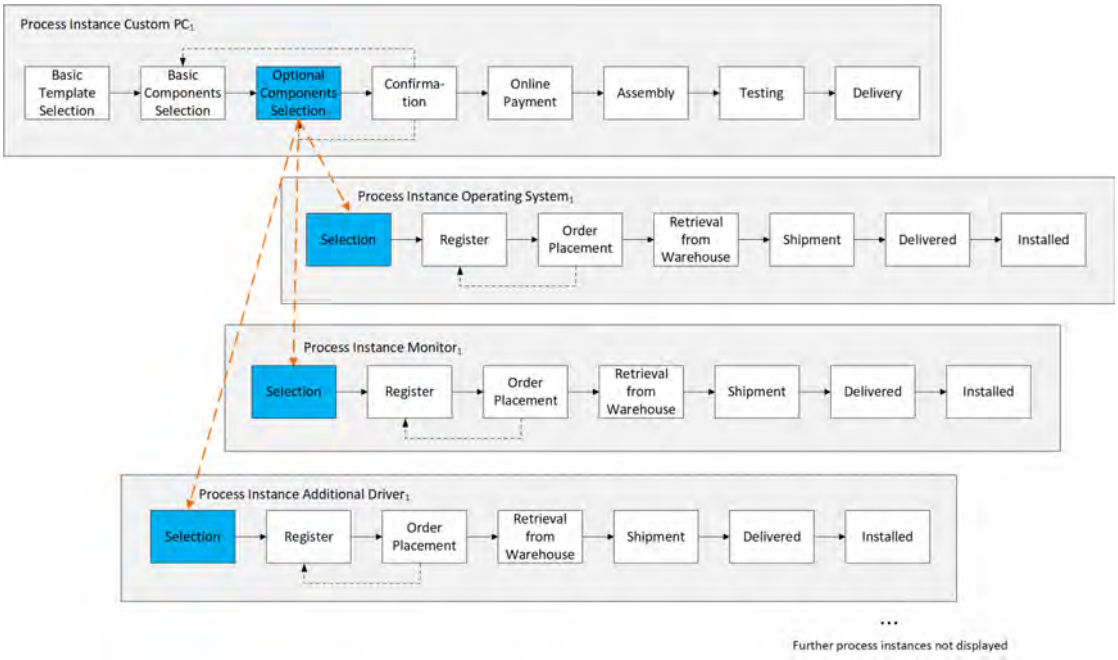
Process Types: Custom-PC, Optional Components

Description: Once the optional components of the custom-PC are configured in the PC configurator by the customer, those may be marked as selected.

Relation Type: $1:n$

Cardinality Restriction: $m^{PC}_{\text{lower}} = m^{PC}_{\text{upper}} = 1; n^{OC}_{\text{lower}} = n^{OC}_{\text{upper}} = n$

Threshold value: $\omega^{PC} = 1$



PCP No.: #3

PCP: PCP 2

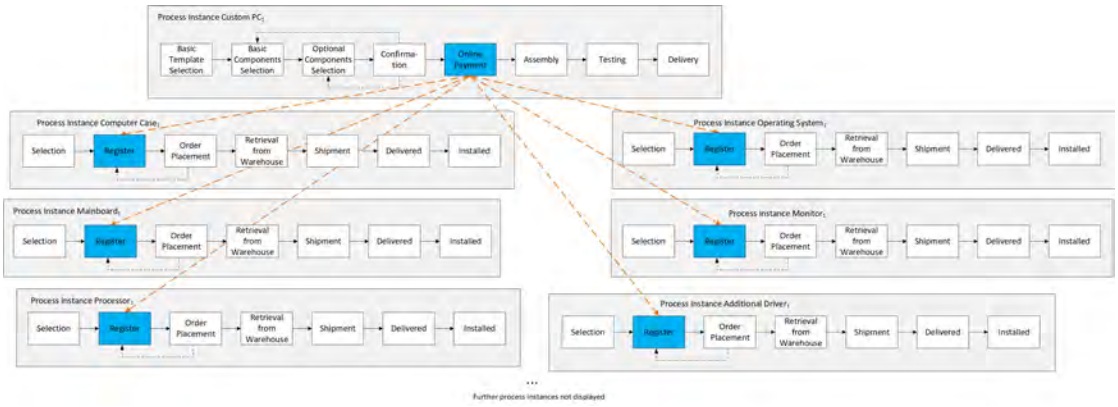
Process Types: Custom-PC, Basic Components, Optional Components

Description: After the online payment of the custom-PC is made, the chosen basic and optional components may be registered by the online retailer.

Relation Type: $1:n$ & $1:n$

Cardinality Restriction: $m^{PC}_{\text{lower}} = m^{PC}_{\text{upper}} = 1$; $n^{BC}_{\text{lower}} = n^{BC}_{\text{upper}} = n$; $n^{OC}_{\text{lower}} = n^{OC}_{\text{upper}} = n$

Threshold value: $\omega^{PC} = 1$



PCP No.: #4

PCP: PCP 2

Process Types: JPO, Basic Components, Optional Components

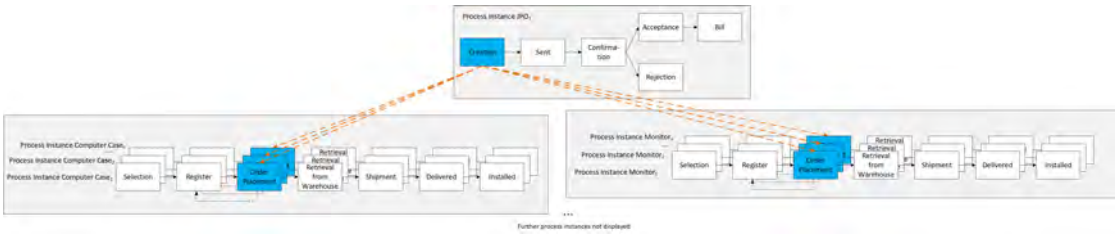
Description: Once a JPO has been created, previously registered basic and optional components offered by the supplier to whom the JPO will be send to, may be placed there.

Relation Type: $1:n$ & $1:n$

Cardinality Restriction: $m^{JPO}_{\text{lower}} = m^{JPO}_{\text{upper}} = 1$; $n^{BC}_{\text{lower}} = n^{BC}_{\text{upper}} = n$;

$n^{OC}_{\text{lower}} = n^{OC}_{\text{upper}} = n$

Threshold value: $\omega^{JPO} = 1$



PCP No.: #5

PCP: PCP 5

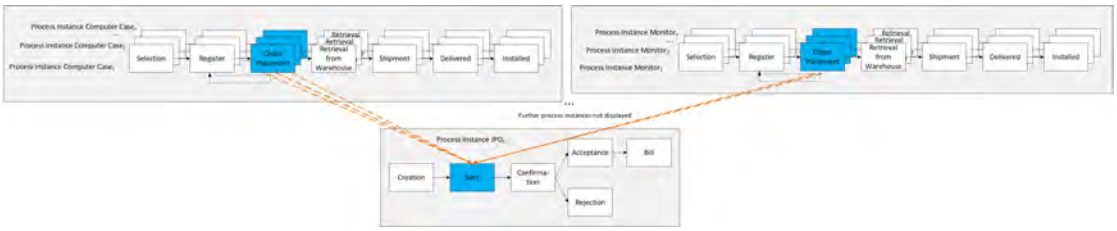
Process Types: Basic Components, Optional Components, JPO

Description: Once placed basic and optional components have reached the required minimum order value, the JPO may be sent.

Relation Type: $m : 1$ & $m : 1$

Cardinality Restriction: $m^{BC}_{\text{lower}} = m^{BC}_{\text{upper}} = m$; $m^{OC}_{\text{lower}} = m^{OC}_{\text{upper}} = m$; $n^{JPO}_{\text{lower}} = n^{JPO}_{\text{upper}} = 1$

Threshold value: $\omega^{BC} = m$; $\omega^{OC} = m$



PCP No.: #6

PCP: PCP 2

Process Types: JPO, Basic Components, Optional Components

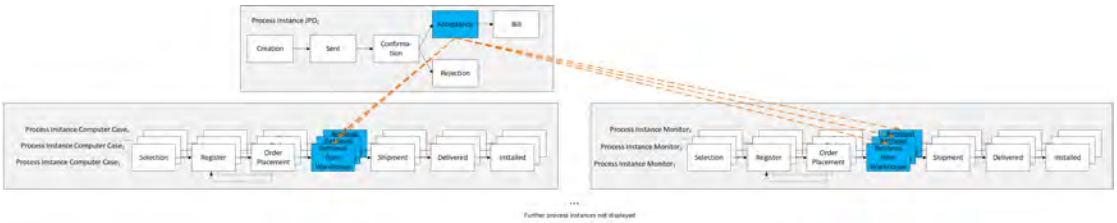
Description: Once the JPO is accepted, the basic and optional components may be retrieved from the supplier's warehouse.

Relation Type: $1:n$ & $1:n$

Cardinality Restriction: $m^{JPO}_{\text{lower}} = m^{JPO}_{\text{upper}} = 1$; $n^{BC}_{\text{lower}} = n^{BC}_{\text{upper}} = n$;

$n^{OC}_{\text{lower}} = n^{OC}_{\text{upper}} = n$

Threshold value: $\omega^{JPO} = 1$



PCP No.: #7

PCP: PCP 5

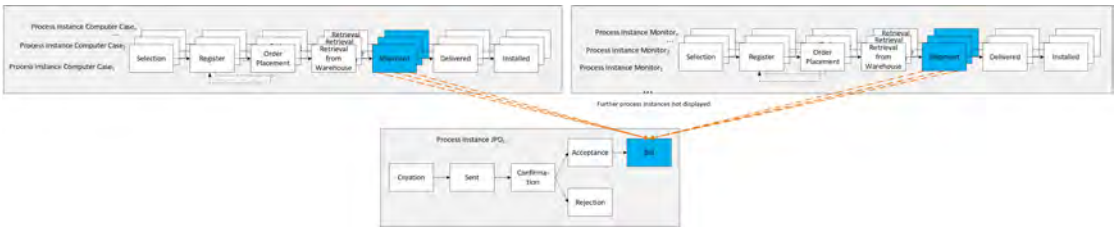
Process Types: Basic Components, Optional Components, JPO

Description: Once the basic and optional components have been shipped, the JPO may be billed to the online retailer.

Relation Type: $m:1$ & $m:1$

Cardinality Restriction: $m^{BC}_{\text{lower}} = m^{BC}_{\text{upper}} = m$; $m^{OC}_{\text{lower}} = m^{OC}_{\text{upper}} = m$; $n^{JPO}_{\text{lower}} = n^{JPO}_{\text{upper}} = 1$

Threshold value: $\omega^{BC} = m$; $\omega^{OC} = m$



PCP No.: #8

PCP: PCP 7

Process Types: JPO, Basic Components, Optional Components

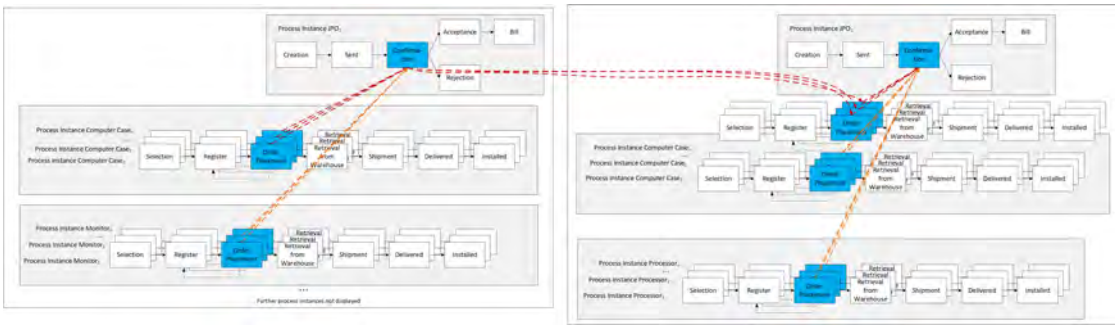
Description: In case of an order rejection, the basic and optional components may be placed into JPOs related to different suppliers who also offer those components.

Relation Type: $1:n$ & $1:n$

Cardinality Restriction: $m^{JPO}_{\text{lower}} = m^{JPO}_{\text{upper}} = 1; n^{BC}_{\text{lower}} = n^{BC}_{\text{upper}} = n;$

$n^{OC}_{\text{lower}} = n^{OC}_{\text{upper}} = n$

Threshold value: $\omega^{JPO} = 1$



PCP No.: #9

PCP: PCP 5

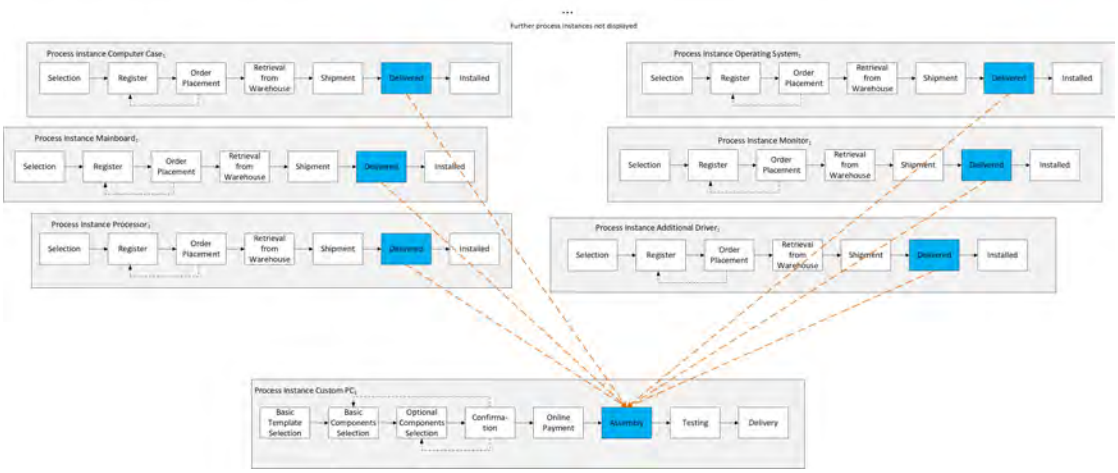
Process Types: Basic Components, Optional Components, Custom-PC

Description: Once all basic and optional components are delivered, the custom-PC may be assembled.

Relation Type: $m:1$ & $m:1$

Cardinality Restriction: $m^{BC}_{\text{lower}} = m^{BC}_{\text{upper}} = m$; $m^{OC}_{\text{lower}} = m^{OC}_{\text{upper}} = m$; $n^{PC}_{\text{lower}} = n^{PC}_{\text{upper}} = 1$

Threshold value: $\omega^{BC} = m$; $\omega^{OC} = m$



PCP No.: #10

PCP: PCP 5

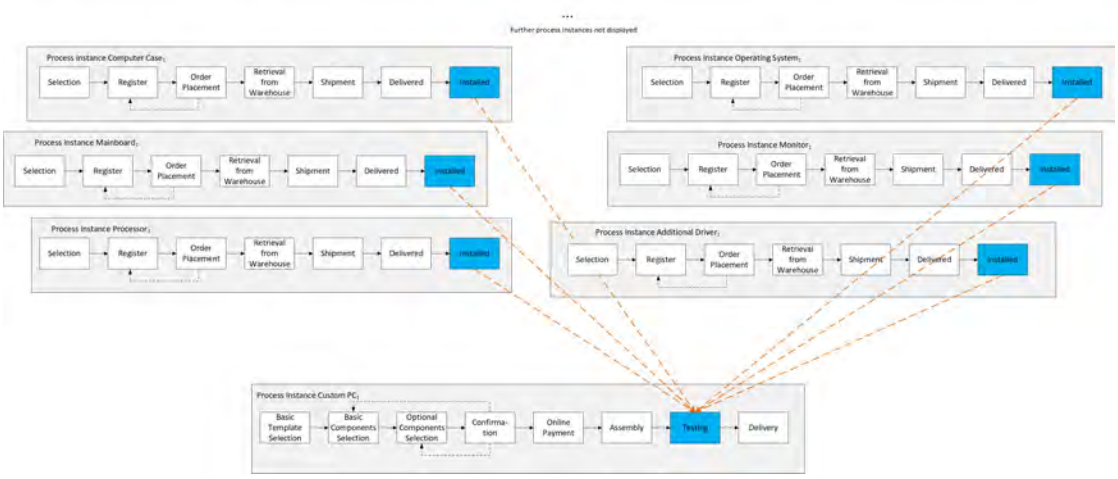
Process Types: Basic Components, Optional Components, Custom-PC

Description: Once all basic and optional components have been installed, the custom-PC may be subject to a rigorous testing.

Relation Type: $m:1$ & $m:1$

Cardinality Restriction: $m^{BC}_{\text{lower}} = m^{BC}_{\text{upper}} = m$; $m^{OC}_{\text{lower}} = m^{OC}_{\text{upper}} = m$; $n^{PC}_{\text{lower}} = n^{PC}_{\text{upper}} = 1$

Threshold value: $\omega^{BC} = m$; $\omega^{OC} = m$



4.4. Conclusions and Statistics

The conclusions and statistics presented in this section are based on the degree of pattern support provided by BPMN and PHILharmonicFlows for the patterns found in Scenario 1 and Scenario 2. The procedure adopted for the evaluation of PCP support is as follows.

First, a general overview of the modelling concepts of each approach is provided by exemplarily describing how BPMN and PHILharmonicFlows capture one PCP, selected from Scenario 1. In this context, only the modelling concepts of each approach that are relevant for understanding the results of the evaluation are mentioned. In subsection 4.4.1, the implementation of the selected PCP is described, thereby referencing the corresponding models of BPMN and PHILharmonicFlows for Scenario 1, set out in the appendix. In order to minimise bias, no attempt, with one exception, was made to optimise the models in favour of PCP support. That is, the models are based on general modelling conventions within the specifications of each approach. The exception concerns modelling Scenario 1 in BPMN with several single pools instead of only two pools, such that process interactions could be modelled and assessed for PCP support. However, this illustration is valid according to the BPMN standard [35].

Second, the degree of support provided by each approach for the selected PCP, was determined based on the evaluation criteria for pattern support stated in Table 4.1. The other PCPs were then evaluated in a similar fashion. The obtained results for the pattern support provided by each approach for Scenario 1 and Scenario 2 are exhibited in subsection 4.4.2. A detailed discussion of the obtained results is provided in the evaluation of each approach in subsections 4.4.3 and 4.4.4. Here, based on the general insights gained from the evaluation, a conclusion about the scope and suitability of PCP support is drawn for each approach.

Finally, threats to validity and interesting findings related to the evaluation are presented in subsections 4.4.5 and 4.4.6.

4.4.1. Models and Exemplary Patterns Implementation

In the following, a general overview of the modelling concepts of each approach is provided by exemplarily describing in detail how BPMN and PHILharmonicFlows graphically represent one PCP from Scenario 1. A detailed description of the modelling concepts of each approach is beyond the scope of this thesis. Though, substantial information is disclosed for generally understanding how Scenario 1 and Scenario 2 have been modelled. Due to the size and number of the models, those are set out in the appendix.

From Scenario 1, the paper selection process of a conference, PCP *1 Simple Succession*, as the basic case of process interaction has been chosen. Since this work focuses on evaluating the degree of support of multiple process interactions, an example of PCP *1*

in a one-to-many relation was chosen. Consequently, PCP # 1 describing that “Once the conference is published, several papers may be submitted at different points in time”, serves as an example.

PCPs abstract process interactions as generally as possible. This generalisation is necessary to account for the heterogeneous definition of interactions in approaches. PCPs thus try to capture the essence of interactions. However, concrete approaches do not necessarily comply to hundred percent to the abstract definition of PCPs. BPMN and PHILharmonicFlows both provide their own solutions for modelling process interactions, each fundamentally different from the other. Therefore, when evaluating PCP support of the approaches, certain pragmatic adaptions are required to account for the discrepancies between approach and the definition of a PCP. A strict interpretation of the PCP definition would result in many *not supported* verdicts (cf. Table 4.1), which for most approaches are unwarranted, as they are caused by the generalisation. This is specially the case for BPMN; the pragmatic adaptions are explicitly pointed out in the description of the models and pattern implementation.

4.4.1.1. BPMN models

The implementation of PCP # 1 from Scenario 1 in BPMN is described hereafter. All the following ◯ reference the BPMN model shown in Figure A.1 in the appendix.

The different participants involved in the scenario are modelled as pools representing private business processes, as in ①. Four pools are used to illustrate the collaboration: *conference organisation*, *authors*, *PB committee* and *PC members*. On the abstract level of PCPs, pools may represent process types. Concerning the number of process instances involved in the scenario, some pools contain a multi-instance marker, indicating participant multiplicity. This is for instance the case for the pools *authors* and *PC members*. Relation types among interacting processes may be interpreted based on multi-instance markers assigned to pools.

Interactions between the different processes rely on electronic message exchanges, which are connected through message flows between pools such as at ②. In this context, a pragmatic adaption is that one message exchange between two pools may not be equivalent to one interaction but it may represent a set, that is, one or more interactions. Based on this interpretation, for instance one message exchange between a retailer and a customer can then be interpreted as several interactions between products (being shipped by the retailer) and an order (being billed to the customer).

Pools and the therein contained *flow elements* (events, gateways, sequence flows, activities, data objects and data associations) describe the behaviour of a process from its creation to its end. Therefore, the elements contained within a pool may be interpreted as the states of the process. In order to point out that multiple instances of an activity are created and executed, an activity may contain a multi-instance marker. Data objects are used to model physical or virtual items such as physical letters, materials

and pdf documents. Data objects are created, manipulated and used during process execution. Data objects representing a collection of data have a multi-instance marker. In BPMN, the notion of states and state changes is not captured explicitly. However, due to the language permissiveness, different solutions for mapping BPMN models to state-based-views have been proposed in the literature. For instance, two examples are mapping BPMN to so-called high-level behavioral diagrams in [10] and mapping BPMN to a process semantics model in [62], based on the mathematical notations Z [11] and CSP [40]. However, those solutions generate vast and complex models. Therefore, in the BPMN models in this thesis, the notion of states within processes as defined by PCPs is endorsed through the use of data objects flowing in and out of flow elements, as in ③. Hence, the states of the process may be explicitly represented. The referenced state of data objects is thereby put into square brackets.

PCP # 1 is expressed at ④ by the pool *conference organisation* sending electronic messages to the pool *authors.* In addition, a signal event is thrown by the *conference organisation,* broadcasting the paper call published online which is caught by all interested *authors*. The processes *conference organisation* and *authors* interact in a one-to-many relationship which is expressed by the multi-instance pool of *authors*. The data objects attached to the electronic messages and the throwing signal event are used to highlight their states. In the PCP context, it can be abstracted that once paper calls are *sent* and/or the signal event is *broadcasted* by the *conference organisation,* several papers are *prepared* and then *submitted* by participating *authors*.

4.4.1.2. PHILharmonicFlows models

The implementation of PCP # 1 from Scenario 1 in PHILharmonicFlows is described hereafter. Additional examples, necessary for understanding the concept of the PHILharmonicFlows framework are also described. All the following ◯ reference the PHILharmonicFlows models shown in the appendix, which are referred to accordingly in the description. The description of the general concepts of PHILharmonicFlows is based on [1, 46, 47, 48, 49].

The types of objects and persons involved in Scenario 1 and existing relations among them are captured in the *data model* in Figure A.2. *Object types* and *relations* are directly derived from the business objects and persons involved in the overall business process described in Scenario 1. Thereby, object types represent small interdependent *processes* with *lifecycle processes* describing the objects' behaviour.[1] As illustrated in the data model, *object types* include *Conference*, *Paper Call*, *Paper*, *Bid*, *Review* and *Meta-Review*. *General Chair*, *Author*, *PC Member* and *PB Member* are *user types,* a special case of an object type representing a person. User types work in conjunction with the authorisation system of PHILharmonicFlows to grant fine-grained permissions, which

[1] Note that, in the context of PHILharmonicFlows the terms *object types* and *processes* are used as synonyms, as objects always have a lifecycle process.

generally allow instantiating and executing object lifecycles. User types thus indicate responsibilities for creating and executing the processes with which they are connected.

Processes are connected to one another through relations of different types similar to the ones found in ER models. For example a *Conference* may be related to one or more *Papers* $(1 : n)$. In turn, one *Paper* may have one or more relations to *Bids* $(1 : n)$. In case of *Reviews,* the relation is restricted to at minimum three and at most five *Reviews* per *Paper* $(3 : 5...1)$. In case of *Meta-Review,* the relation is restricted to one *Review* per *Paper* $(1 : 1)$. The same applies for *Conference* and *Paper Call* $(1 : 1)$.

Established relations between processes at design-time indicate process dependencies at run-time. This means that, at a specific point in time, the execution of a process instance depends on the execution status of a related process instance. In this context, process dependencies are expressed in the data model through directed edges between object types. The direction of the relation classifies processes into *lower-level* and *higher-level processes*; i.e. the former is the child process and the latter the parent process. For example, *Paper* is a higher-level process of *Bids*, *Reviews* and *Meta-Reviews.* Directed edges further allow to express common process relations, e.g. *Paper Call* and *Paper* are both related to *Conference,* a common higher-level process. Furthermore, also transitive relations are taken into account in the data model. An example are *Review* and *Conference* which are not directly related, but transitively via a path of relations. Transitive relations indicate that, dependencies exist between processes which are not directly connected. Simply put, at a specific point in time, the execution of a *Conference* instance depends on the execution status of its transitively related *Review* instances. It can be deduced from the data model that one-to-many relations are directed, where 1 is assigned to the higher-level process and n to the lower-level process. Many-to-many relationships may be broken down into several one-to-many relations.

Part of the data model is also a set of attributes assigned to each object type. Because of space reasons, those have been omitted in the data models of Scenario 1 and Scenario 2 illustrated in Figure A.2 and Figure A.7. The attribute types are elementary and describe the data held in the corresponding object types. An example attribute type is *Title*, a String defined in the *Paper* object type. Attribute types assigned in the data model are relevant for describing an object's behaviour in the corresponding object's lifecycle process.

Each object and user type is required to have a *lifecycle process*, which is described by a directed acyclic graph. The lifecycle process may be seen as a *state-based-view*, that consists of different *states* describing the object's behaviour at design-time. A lifecycle process has one start state and at least one end state. In this context, attribute types defined in the data model are grouped into the states of an object's lifecycle process. States are connected to one another by *transitions*. In object lifecycles only one state may be active at any point in time. At run-time, the transition of an object from one state to the next one is based on the assignment of attribute values by authorised users, i.e. *user types* with which the *object type* is related in the data model. Lifecycle processes are thereby data-driven, specifying the default order in which attribute values need to be

present for a process to progress. Backwards transitions allow returning to a previous state in case that an attribute value needs to be corrected. The operational semantics of lifecycle processes enables flexibility by allowing for data values to be entered at any point in time while ensuring a correct process execution.

The lifecycle processes, i.e. the state-based-views of the object types found in Scenario 1 and Scenario 2 are illustrated in Figure A.3 and Figure A.8. In those representations, only the states of the object lifecycle processes are illustrated; attribute types have been omitted for reasons of simplicity.

In PHILharmonicFlows, the data model and lifecycle processes can be seen as the preceding steps to arrive at an expressive overall business process. At run-time, processes run, in general, independently from one another. Occasionally for the overall business process to advance, the progress of an object type may depend on the execution status of another object type. On this account, the behaviour of the involved processes needs to be coordinated, i.e. processes need to interact with one another. *Coordination processes* are thereby used to model process interactions.

Based on the defined relations in the data model, process interactions are modelled in coordination processes by coordinating the lifecycle processes of related object types. Coordination processes rely thereby on *semantic relationships,* which represent dependency constraints in one-to-many and many-to-many relationships that may exist between processes. An example for such a dependency constraint is the condition stated by PCP # 1 that “Once the *Conference* is published, several *Papers* may be submitted at different points in time”. Semantic relationships are based on five patterns that can be combined in various ways to specify complex process dependencies. Table 4.2, taken from [47], gives an overview over semantic relationships.

In coordination processes, semantic relationships can be automatically derived from the underlying data model. Strictly speaking, this means that, semantic relationships may only be established between processes if a path of relations exists between these processes.

Tabelle 4.2.: 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.

Coordination processes allow to only expose the interactions which are useful for understanding the overall business process. Coordination processes take into account that interacting process instances may be executed asynchronously. This means that, a coordination process intervenes only when necessary, at specific points during the execution of a process instance, impacting its execution as little as possible.

Complex scenarios, such as Scenario 1 and Scenario 2, may be broken down into several coordination processes or modelled as one. Coordination processes may be modelled based on the defined process hierarchy in the underlying data model, i.e. taking into account higher-level and lower-level process relations.

By way of example, Scenario 1 is illustrated with two coordination processes. This representation is based on an existing process hierarchy between *Conference* and *Paper* (cf. Figure A.2). The coordination process *Conference* in Figure A.4 illustrates the overall business process from a higher perspective, describing the interactions between *Conference* and *Paper,* thus abstracting from details related to the paper selection process. In comparison, the coordination process *Paper* in Figure A.5 describes the individual stages that take place from the creation of a paper till the final selection assessment. Both coordination processes convey a picture of the overall business process from different perspectives while revealing how they are interrelated.

Another example concerns Scenario 2, which is illustrated with two coordination processes. This representation is based on the fact that a process hierarchy between

Custom-PC and *Joint Purchase Order* does not exist, also neither through a common higher-level process (cf. Figure A.7). This implies that they are independent from one another. Both processes are only indirectly related through the basic and optional components such that interactions between *Custom-PC* and *Joint Purchase Order* do not occur. Interactions occur only between *Custom-PC* and the different components, modelled in the coordination process *Custom-PC* (cf. Figure A.9), and between *Joint Purchase Order* and the different components, modelled in the coordination process *Joint Purchase Order* (cf. Figure A.10).

Coordination processes allow to model coordination constraints for multiple interrelated process instances while using only three modelling elements: *coordination steps*, *coordination transitions* and *ports*. Coordination processes are represented as a directed graph. In the following, it is exemplarily explained how PCP # 1 from Scenario 1 is expressed with the coordination process *Conference*. All the following ◯ reference this coordination process shown in Figure A.4. The operational semantics of the modelling elements are thereby briefly described.

Coordination steps are the building blocks of the model, referring an object type as well as one of its states. The coordination step is addressed in the form *object type:state*, e.g. ① *Conference*:*Preparation.* A coordination step can be interpreted as a container of process instances of the referenced object type. Coordination steps thus provide an abstract way to represent multiple process instances at design-time, which are then enacted at run-time.

A *coordination transition* is a directed edge connecting coordination steps with one another. Based on the direction of the edge, coordination steps are differentiated into *source* and *target coordination steps.* In this context, both source and target coordination steps reference an object type of the data model. Coordination transitions recognise the relations between higher-level and lower-level processes defined in the data model. To put it concisely: By creating coordination transitions between coordination steps, semantic relationships are automatically derived based on two things, the relations defined in the data model, and the referenced lifecycle states of the objects being coordinated. For example at ②, connecting *Conference*:*Published* to *Paper Call*:*Announcement* constitutes a top-down relationship indicating that for a paper call to be announced, first a conference needs to be published.

At run-time, coordination processes enforce semantic relationships based on the execution status of the objects between which semantic relationships are defined. In other words, an object's process execution may progress or halt based on whether conditions captured by semantic relationships are currently satisfied or not. Taking PCP # 1 as an example, by connecting *Paper Call*:*Announcement* to *Paper*:*Submitted* at ③ a transverse relationship is created. The semantic relationship between these coordination steps enforces that a *Paper Call* must reach state *Announcement* before any *Paper* may be in state *Submitted*. Once a particular *Paper Call* reaches state *Announcement* and the state becomes active, the transverse semantic relationship becomes enabled and subsequently allows any number of *Papers* to be submitted, at different points in time.

So far, it follows from all the foregoing descriptions that, PCP # 1 from Scenario 1 can be modelled with the PHILharmonicFlows framework.

A coordination process graph is acyclic and connected, implying that for a semantic relationship to be enabled it is required that all predecessor semantic relationships must have been enabled. This means that, for example enabling the bottom-up semantic relationship of *Paper*:*Initial Check Rejected* with *Conference*:*Not Participant* requires that the top-down semantic relationship *Conference*:*Published* with *Paper Call*:*Announcement* has already been enabled. Furthermore, as it may be generally deduced from the coordination processes, the start and end coordination steps of a coordination process must reference the start and end states of the lifecycle of the coordinated object type. In this way, a proper start and completion of the coordination process is ensured.

As it has been mentioned above, for more complex coordination constraints to be expressed, multiple semantic relationships may need to be combined. This is enabled through the concept of *ports*, which are incorporated in coordination processes. *Ports* are attached to a coordination step ④, such that coordination transitions do not directly target a coordination step but the port attached to it. Except from the start coordination step, any coordination step must have one or more ports. Ports allow realising AND- and OR-semantics when combining semantic relationships. Connecting multiple coordination transitions to the same port corresponds to AND-semantics. This means that, for the port to become enabled, which in turn activates the attached coordination step, all semantic relationships attached to the incoming coordination transition must be enabled. Connecting coordination transitions to different ports attached to a coordination step corresponds to OR-semantics. Hence, for the coordination step to be activated, at least one port must be enabled. Moreover, AND-and OR-semantics can be freely combined to create advanced boolean expressions.

From the provided description, it is recognisable that the concept of AND-and OR-semantics provided by semantic relationships allows to model more complex process interactions as the one described by PCP # 1 in Scenario 1.

An example for a more advanced pattern that can be properly modelled with AND-semantics is PCP # 10 from Scenario 2. The implementation of this pattern is found in the coordination process *Custom-PC* in Figure A.9. The coordination step *Custom-PC:Testing* at ⑤ has one port with thirteen incoming coordination transitions. Therefore, for the custom-PC to be subject to a rigorous testing, all conditions represented by the semantic relationships need to be fulfilled. The bottom-up semantic relationships outgoing from the source coordination steps referencing basic and optional components require a sufficient number of each component (e.g. at least one) to have reached state *Installed* before the custom-PC may be tested. The exact number of each component required to be in state *Installed* is thereby a design-choice that can be configured with an expression framework provided by coordination processes.

One example for an OR-semantics construct, which allows to model two patterns from Scenario 2 at once, is found in the coordination process *Paper* at ⑥ in Figure A.5. Rejecting a *Paper* may be achieved in two different ways. First, the *Meta-Review*

corresponding to a *Paper* endorses an immediate rejection; thereby PCP # 5 is supported. Second, during the selection assessment of the set of available *Papers,* a rejection of the *Paper* is endorsed, and the *Paper* is rejected then, which provides support for PCP # 6. In the first case the corresponding semantic relationship is bottom-up while in the second case it is a self semantic relationship, both connect to two different ports of the coordination step *Paper:Rejected*. In a nutshell, the OR-semantics allows rejecting the *Paper* in either case.

In order to express more complex coordination constraints, semantic relationships provide *configuration options*, which are based on an *expression framework*. Boolean, arithmetic functions, constants and variables based on process data are part of the expression framework. The expression framework reflects the process context to be evaluated while keeping the expressions simple. This is achieved by relying on specialised counting functions. Such functions count process instances, which are represented by the source and target coordination steps between which a semantic relationship is created. Counting functions take particularly into account the active state of process instances at run-time. For example, depending on whether a particular state is active, has been active, has not yet been active, or has been skipped, the configured semantic relationship may be enabled or not. In addition, the expression framework allows realising a NOT boolean operator.[2]

To name an example from Scenario 1, configuration options enable a process modeller to explicitly specify that at least three *Reviews* have to be submitted per *Paper* for a *Meta-Review* to be prepared. On a side note, this coordination constraint is contained in PCP # 4. The corresponding semantic relationship has been annotated with its respective coordination expression at ⑦ in Figure A.5. The expression implemented in the model is:

$$(\#SourceIn \geq 3) \| (\#SourceIn = \#AllSource)\,\&\&\,(\#SourceIn > 0).$$

This expression covers more than the constraint demanded by PCP # 4. It expresses that, at run-time, in the context of one *Paper*, a *Meta-Review* may be prepared as soon as three *Reviews* reach the state *Closed,* or based on the existing *Reviews* that are in the state *Closed,* whereby at least one must reach this state.

In case that, a modeller has not configured a semantic relationship, it defaults to the expression $(\#SourceIn = \#AllSource)\,\&\&\,(\#SourceIn > 0)$. This implies that, the referenced state of the source coordination step must be active in all process instances. The default expression is concretised by means of an example, the coordination process *Joint Purchase Order* in Figure A.10. The bottom-up semantic relationships between the source coordination steps referencing several *basic components* in state *Shipment* and the target coordination step referencing *Joint Purchase Order:Billed,* at ⑧ are set to default. This means that, in the context of one *Joint Purchase Order,* for the semantic relationship to become enabled, the instances that are active within each

[2] A detailed description on the expression framework implemented in coordination processes is beyond the scope of this thesis and can be found in [47].

source coordination step, must correspond to all existing instances of each source coordination step.

In the coordination models of Scenario 1 and Scenario 2 found in the appendix, only the described expressions are shown. The remaining semantic relationships are set to default and therefore not illustrated.

4.4.2. Overview of Evaluation Results

The scenarios modelled with BPMN and PHILharmonicFlows were chosen to cover as many different PCPs as possible. Each scenario exhibits different PCPs, allowing to examine the degree to which the selected modelling approaches can adequately model PCPs. The degree of pattern support of both modelling approaches is shown in Table 4.3 for Scenario 1 and in Table 4.4 for Scenario 2. The results are based on the evaluation criteria *relation type*, *cardinality restriction* and *threshold value* ω, presented in Table 4.1. Note that the modelling approaches show a different degree of pattern support. Nonetheless, this evaluation is not an overall quality assessment of the individual approaches.

Tabelle 4.3.: Overview of Pattern Support for Scenario 1

Scenario 1: Paper selection process of a conference			
#	PCP	BPMN	PHILharmonicFlows
1.	PCP *1*	**+/-**	**+**
2.	PCP *3*	**+/-**	**+**
3.	PCP *1*	**+/-**	**+**
4.	PCP *1*	**+/-**	**+**
5.	PCP *1*	-	**+**
6.	PCP *1*	**+/-**	**+**
7.	PCP *1*	**+/-**	**+**
8.	PCP *1*	**+/-**	**+**
Direct support			8/8
Partial support		7/8	
No support		1/8	

Tabelle 4.4.: Overview of Pattern Support for Scenario 2

Scenario 2: Built-to-Order process			
#	PCP	BPMN	PHILharmonicFlows
1.	PCP *2*	**+/-**	**+**
2.	PCP *4*	**+/-**	**+**
3.	PCP *2*	**+/-**	**+**
4.	PCP *2*	-	**+**
5.	PCP *5*	**+/-**	**+/-**
6.	PCP *2*	-	**+**
7.	PCP *5*	**+/-**	**+**
8.	PCP *7*	**+/-**	**+**
9.	PCP *5*	**+/-**	**+**
10.	PCP *5*	-	**+**
Direct support			9/10
Partial support		7/10	1/10
No support		3/10	

The obtained results in Table 4.3 for Scenario 1 and in Table 4.4 for Scenario 2 are discussed in the evaluation of each approach in subsections 4.4.3 *BPMN Evaluation* and 4.4.4 *PHILharmonicFlows Evaluation*. Moreover, encountered problems and corresponding reasons are mentioned. The obtained results are based on the degree of support provided by each approach for PCP # 1 from Scenario 1. The other PCPs have been evaluated in a similar fashion, giving the results in Table 4.3 and Table 4.4.

4.4.3. BPMN Evaluation

In subsection 4.4.1 the implementation of PCP # 1 from Scenario 1 was described. The degree of support provided by BPMN for PCP # 1 as shown in Table 4.3, is assessed as partial for the following reasons. The ◯ reference the BPMN model in Figure A.1 in the appendix.

Concerning the feature *relation type*, one process, the *conference organisation,* is related to several processes, *the authors* which are modelled as a multi-instance pool. On an abstract level, it is identifiable that the conference published by the *conference organisation* is related to several papers created by different *authors*. Modelling the relation type one-to-many, where *many* remains an abstract value n as stated by the assessed PCP is thus supported.

Regarding the feature *cardinality restriction*, BPMN provides the option of assigning properties to certain flow elements and organisational elements (e.g. pools, swimlanes). Yet, property elements are not visually displayed on a process diagram but contained within a flow or organisational element. For instance, a minimum and maximum value defining the number of participants within a pool can be assigned. However, the values assigned have to be specific numbers, implying that the number of process instances have to be known at design-time. Though, this is not always the case. For instance, the number of authors that may submit papers for a conference is unknown at design-time. By assigning *cardinality restrictions* to pools, BPMN explicitly specifies the number of participants within one process. Contrary to PCPs, in BPMN the number of participants within one process that can be related to participants of another process is implicitly stated. In other words, specifying the number of instances of one process type that can be related to instances of another process type as defined by PCPs is not explicitly foreseen. However, this fact is problematic when one process is related to several processes where each relation is of a different type. In such a case, the number of process instances related to one another is ambiguous. Based on the findings hereby provided, it can be generally concluded that the degree of support provided by BPMN for the feature *cardinality restriction* is *partial.*

Relating to the feature *threshold value* ω, explicitly modelling a number of process instances that need to be synchronised into a specific execution state such that related process instances can be executed is only possible through workarounds. BPMN it not primarily a data flow language; modelling the data perspective is considered optional by BPM experts and therefore often omitted for reasons of complexity reduction. According to the BPMN standard [35], data objects do not have any direct effect on the sequence flow or message flow of processes. However, modelling data objects indicating their states provides a possibility for depicting the *threshold value* ω to some extent. By modelling the multi-instance data object *<paper call [sent]>* attached to the message send activity at ⑤, it is visualised that the *conference organisation* interacts with the *authors* based on the former process reaching a specific execution state. However, the BPMN standard [35] leaves the structure of data objects underspecified. BPMN experts differ in opinions whether a data object that is a collection of data represents multiple instances of an object or just one object instance with a list of fields cf.[7, 28, 41]. Therefore, contrary to pools, data objects do not have a property element for determining the number of data object instances that can be created. For the evaluation of PCP support, it is assumed that a multi-instance data object represents multiple object instances. However, the number of object instances that need to reach a specific execution state for an interaction to take place still remains ambiguous. A solution for this problem could be using textual annotations attached to the data objects, indicating the *threshold value* ω needed for process interactions. Note that, textual annotations are not binding, they are merely a mechanism to provide additional text information in a BPMN model [35]. Based on the established evaluation method in section 4.1 data objects as well as textual annotations represent workarounds for picturing the feature

threshold value ω. It can be therefore stated that, in general BPMN provides *partial* support for the feature *threshold value* ω.

All aspects considered, based on the evaluation criteria for pattern support stated in Table 4.1, partial support is assigned to BPMN for PCP # 1.

A further substantial information for understanding the results stated in Table 4.3 is the evaluation of PCP # 5 from Scenario 1. This pattern describes that "Once all meta-reviews for the whole set of available papers is submitted, the final set of papers may be selected". As it can be seen at ⑥, the activities *submit meta-review* and *select final papers* and their corresponding data objects are contained in the pool *PB committee* such that there is no interaction. As a result, in this case a process interaction as defined by PCP # 5 cannot be modelled and is thus, not supported. In the evaluation overview in Table 4.3 and Table 4.4, for some cases BPMN has been evaluated as providing no pattern support. In general, this results from the fact that, analogous to the mentioned example, interactions between pools capturing the assessed patterns are missing.

To mention one further significant finding, a more complex pattern, PCP *2 Concurrent Succession*, is briefly assessed hereinafter. From scenario 2, the BTO process for a custom-PC, PCP # 3 has been chosen for assessment. PCP # 3 describes that "After the online payment of the custom-PC, the chosen basic and optional components may be registered by the online retailer". The implementation of the pattern is shown in the BPMN model in Figure A.6 in the appendix; the following ◯ reference this model. The degree of support provided by BPMN for PCP # 3 is assessed as *partial*.

Concerning the feature *relation type,* according to PCP # 3 one custom-PC is related to several components such that several interactions may take place. In BPMN, process interactions occur only between collaboration participants via message exchanges. Therefore, a pragmatic adaption is made, such that the individual process interactions described by PCP # 3 are abstracted and modelled as one message exchange between the two processes *customer* and *online retailer*. One message exchange may thus represent one or more interactions. Thereby, on an abstract level, it is identifiable that one custom-PC is related to several components. This is represented through the message exchange between the pools *customer* and *online retailer* to which the data object *<custom-PC-order [paid]>* is attached at ⑦. Based on the custom-PC being paid by the *customer,* several components are registered by the *online retailer.* This is highlighted with the multi-instance data object *<components [registered]>* found in the pool *online retailer* ⑧. Modelling the relation type one-to-many between custom-PC and components, where *many* remains an abstract value n as stated by the assessed PCP is supported in a roundabout way.

Regarding the feature *cardinality restriction*, as already mentioned, in BPMN the number of process instances related to one another remains ambiguous. Therefore, BPMN generally provides *partial* support to the feature *cardinality restriction*. Since the feature *threshold value* ω may not be explicitly assigned through formal constructs of BPMN, here again *partial* support is awarded.

All in all, evaluating the support of multiple process interactions as defined by PCP *2 Concurrent Succession* requires pragmatic adaptions to account for the discrepancies between approaches and the definition of a PCP. This is done by bundling several interactions into one message exchange. In fact this means that, process interactions of the type 1:n as stated by some of the patterns assessed, could be contemplated and evaluated as such in BPMN. Otherwise, in most cases of Scenario 2, process interactions of the type 1:n, would correspond to process interactions of the type 1:1 in BPMN, thus showing less PCP support.

The other PCPs have been evaluated in a similar fashion, leading to the presented results in Table 4.3 and Table 4.4.

4.4.4. PHILharmonicFlows Evaluation

In subsection 4.4.1 the implementation of PCP # 1 from Scenario 1, among further examples, was described. As shown in Table 4.3 and Table 4.4, PHILharmonicFlows has been assessed as providing a full degree of support for almost all of the patterns found in Scenario 1 and Scenario 2. The only exception is PCP # 5 from Scenario 2, assessed as partially supported. This exception is further specified after accounting for the overall full support awarded to PHILharmonicFlows.

In general it can be stated that, the overall concept of PHILharmonicFlows feats neatly with the essence of PCPs for the following reasons. The abstract notion of processes with a corresponding lifecycle process on which PCPs are build upon, is derived from the object-aware approach, on which also PHILharmonicFlows is based. On the abstract level of PCPs, object types may therefore represent process types.

Concerning the feature *relation type,* the data model allows to capture processes and the relations between them at design-time. In this context, process relations can be explicitly defined in an abstract manner, e.g. $1 : n$. Many-to-many relations are inherently supported by breaking them down into several one-to-many relations. Regarding the feature *cardinality restriction,* the data model further allows assigning cardinality constraints to process relations. With the data model, process relation awareness is captured at design-time and enforced at run-time. It can be generally concluded that PHILharmonicFlows provides *full* support for the features *relation type* and *cardinality restriction.*

Relating to the feature *threshold value* ω , it can be generally stated that, this feature is *fully* supported by PHILharmonicFlows through the concept of coordination processes modelled at design-time. Coordination processes capture complex process interactions by specifying coordination constraints and enforcing them at run-time. Dependency constraints among process instances are thereby captured through semantic relationships, which can be configured with a context-aware expression framework. Configurating semantic relationships allows to explicitly set *threshold values* ω such that interactions

occur based on the execution states of the interacting process instances, taking into account asynchronous process execution.

In the main, the PHILharmonicFlows framework supports the characteristic features of PCPs. However, PCP # 5 from Scenario 2, as the only exception, has been assessed as being *partially* supported. This results from the fact that, this patterns displays a higher degree of complexity, relatively to the other ones. PCP # 5 describes that "Once placed basic and optional components have reached the required order value, the joint purchase order may be sent". The implementation of this pattern is illustrated in the coordination process *Joint Purchase Order* shown in Figure A.10. At ⑨ the target coordination step *Joint Purchase Order:Sent* has one port with thirteen incoming transitions outgoing from the source coordination steps referencing basic and optional components in the state *Order Placement.* However, each one of the bottom-up semantic relationships outgoing from the source coordination steps, require a specific value to be reached for the semantic relationship to be enabled. This implies that, the configured expressions must be able to aggregate attribute values assigned to the source coordination steps, in this case the prices, such that based on predefined values being reached, i.e. minimum values, the semantic relationships are enabled. Aggregate functions are not yet part of the expression framework of PHILharmonicFlows. However, as mentioned in [47], the expression framework is not limited to the existing functions and may be expanded in the future such that more complex constraints may be expressed. Since the essentials of PCP # 5 can be modelled, i.e. interactions based on execution states, partial support was awarded.

Finally, one further substantial information concerns the support provided by PHILharmonicFlows for PCP # 8 from Scenario 2. This pattern represents PCP 7 *Reassignment,* the special case of the PCP catalogue, involving the rerouting of process instances. PCP # 8 states that "In case of an order rejection, the basic and optional components are placed into joint purchase orders related to different suppliers who also offer those components". The implementation of this pattern is implicitly enabled, in particular, by how semantic relationships coordinate the processing of object types with the processing of other object types based on their execution status. Though, backwards transitions in lifecycles processes play thereby a major role. PCP # 8 is implemented in the process *Joint Purchase Order* shown in Figure A.10; described in the following. Referencing PCP # 5 from Scenario 2, described above; provided that a JPO instance reaches the state *Sent* at ⑨*,* the JPO may then either proceed to state *Accepted* or to state *Rejected*. Accepting a JPO enables the top-down semantic relations between JPO and the components that may be part of the JPO at ⑩. This means that, a component instance may then transit from the state *Order Placement* to the state *Confirmation* in its corresponding lifecycle process (cf. Figure A.8). By looking at the defined lifecycle process for components and the coordination process *Joint Purchase Order* in Figure A.10, it can be derived that: In case that a JPO is rejected*,* a component instance cannot transit to the next state *Confirmation*, thus remaining in the state *Order Placement.* Based on the design-time information, the run-time environment of the PHILharmonicFlows framework enables tracking the execution progress of individual process instances as

well as their relations. This means that, authorised users may trace component instances that are “stuck” in the state *Order Placement* due to a rejected JPO. Then, by enabling the backwards transition in the lifecycle process of a component, a state change from *Order Placement* to state *Registration* is possible. The component instance may then be placed in another JPO. By resetting the state of the process instance component, the bottom-up semantic relation between the coordination step referencing the component and the coordination step referencing the JPO is erased. In other words, the process instance component is no longer related to that specific JPO. The run-time environment of the PHILharmonicFlows framework enables to obtain accurate information about the execution state of process instances and their relations at any point in time.

4.4.5. Threats to Validity

In the following, factors that may call into question the evaluation results of pattern support provided by BPMN and PHILharmonicFlows are discussed. These factors are denoted as threats to validity.

With respect to BPMN, a threat to validity poses the fact that modelling process interactions depends on the modeller’s perception of the overall business processes. To be more specific, it is conceivable to model Scenario 1 with two pools only. One pool could represent the *paper selection process of a conference* and be sub-partitioned into the lanes *conference organisation, program board* and *program committee.* This representation is based on the fact that according to the BPMN standard [35], the meaning of the lanes is up to the modeller. Lanes are commonly used for allocating activities to roles or systems within one process. In that sense, the mentioned lanes could be seen as roles of one process. The other pool could represent the *authors*. Based on how interactions are defined in BPMN, evaluating the PCP support required to model the processes involved as single pools. Otherwise, there would not have been many interactions to assess. Furthermore, relationship types between distinct lanes are not supported in BPMN. Therefore, in order to capture the types of relations among interacting processes, those were modelled as single pools. However, the decision of modelling Scenario 1 with four processes instead of two represents potential bias in favor of BPMN.

Another threat to validity is that, for the evaluation, PCPs were applied with some room for interpretation. This stems from the fact that, even tough PCPs abstract process interactions as generally as possible, concrete approaches such as BPMN do not necessarily comply to hundred percent to this abstract definition. Therefore, when evaluating PCP support of the approaches, certain pragmatic adaptions were required to account for the discrepancies between approaches and the definition of a PCP. In this context, for the assessment of BPMN one message exchange between two pools was not perceived as being equivalent to one interaction but to a set, that is, one or more interactions. Hence, several interactions may be bundled into one message exchange. As a result, process interactions of the type 1:n as stated by some of the patterns assessed, could be apprehended as such in BPMN. Otherwise, in most

cases, process interactions of the type 1:n, would correspond to process interactions of the type 1:1 in BPMN, thus complicating a comparison. It is worth noting that, the pragmatic adaptions taken are not completely arbitrary. The BPMN standard [35] does not concretely specify the ratio between messages, message flows and interactions. In the context of collaborations, it merely mentions that the interaction between two pools is shown through message flows. Then again, in the context of choreographies it is mentioned that one or more message flows represent interaction(s) between two participants. Consequently, it is underspecified how many messages correspond to an interaction and vice versa.

All in all, the pragmatic adaptions concerning process interactions represent an interpretation in favour of BPMN. If the PCPs would have been applied too dogmatically, the evaluation of BPMN would show less support for PCPs, due to the rigidity in the application of the PCPs. However, a strict evaluation leading to a worse performance of BPMN in terms of PCP support would be also legitimate.

Relating to PHILharmonicFlows, no pragmatic adaptions were required as the interactions concept of PHILharmonicFlows rather coincides with the definition of PCPs. Yet, this may result in a more strict evaluation of PHILharmonicFlows as PCPs were applied as defined. An example is the partial support awarded for PCP # 5 from Scenario 2. Even tough, the main essence of the pattern is supported, on the whole only partial support was awarded, as the assessment delved into technical details by examining the expression framework.

4.4.6. Interesting Findings

In the context of the evaluation, interesting findings were identified, which are reported in the following.

An interesting finding is that in Scenario 1, the basic case of process interactions, PCP 1, accounts for 7 of the 8 patterns present in the scenario (87,5%). In Scenario 2, PCP 2 and PCP 5, each account for 4 of the 10 patterns present in the scenario (40%). Scenario 1 comprises five process types while Scenario 2 comprises up to fifteen process types (depending on how many optional components are selected). It could be assumed that, in scenarios involving several process types, as it is the case in Scenario 2, more advanced PCPs can be found. However, this assumption would have to be investigated in future work by gathering numerous different process scenarios.

Another interesting finding is that, PCP 6 is the only pattern that is not part of any of the scenarios modelled, such that its support could not be evaluated. The scenarios modelled with BPMN and PHILharmonicFlows were chosen based on literature with focus on multiple interacting process instances. The intention was thereby to cover as many different PCPs as possible. However, since there is not much literature with focus on multiple process interactions, it could not be guaranteed that the evaluation would cover all PCPs of the current catalogue. A possible approach for future investigations

on PCP support could be to conduct an extensive SLR with focus on multiple process interactions. Hence, a much larger data basis would be available for a systematic selection of scenarios involving multiple process interactions.

One further interesting finding is that in PHILharmonicFlows with one coordination construct, i.e. a connection of coordination steps, coordination transitions and ports, it is possible to model more than one PCP at once. As already described in subsection 4.4.1, this is the case for PCP # 5 and PCP # 6 from Scenario 2, which can be both captured with one OR-semantics construct.

Finally, one interesting finding related to Scenario 1 is the review-phase, where reviews assigned to a PC member may be reassigned to a different PC member in case that the former cannot meet the submission deadline. The described reassignment process does not comply with PCP 7 from the Pattern Catalogue as it involves the rerouting of single process instances from one user to another. PCPs do not consider user as processes, therefore the described rerouting was not considered as a pattern for assessment. However, since PHILharmonicFlows differentiates between object types and user types, the routing of reviews from one PC member to another could be considered as a special case of PCP 7, which has been adapted for the approach. The reassignment of reviews could be implemented in PHILharmonicFlows by defining a backwards transition from the state *Paper Assessment* to the state *Assignment* in the lifecycle process of *Review* (cf. Figure A.3 in the appendix). Authorised users, for example PB members, could then reset the state of a *Review* and assign it to a different PC member.

The evaluation performed in this thesis provided an initial insight in interesting findings. In the future, a more extensive evaluation involving more scenarios, such that more PCPs appear in different ratios, is likely to generate new insights.

5
Related Work

This chapter discusses other pattern catalogues describing process interactions that take into account relationship types. In addition, modelling approaches that take multiple process interactions into consideration are also briefly presented.

Service Interaction Patterns (SIPs) [4, 55] comprise a collection of thirteen patterns focused on interactions between organisation-centric processes. Interactions between processes occur on a service-oriented basis, where each business process represents a service provider. Interactions between business processes are based on message exchanges between activities within those processes. SIPs describe interactions between processes based on the number of participants involved and the number of messages exchanged in a business transaction. However, many-to-many relations are not considered in the pattern catalogue, merely one-to-many and many-to-one relations are taken into account.

Correlation Patterns [3] are a collection of eighteen patterns that also describe interactions between services through message exchanges. The focus lies on the matching of incoming and outgoing messages between business processes, termed as correlation. Correlation patterns are based on three main concepts, events, conversations and process instances, which are grouped into correlation mechanisms. The scenarios described, in which correlations might occur, take to some extent multiplicity of process instances into account. Analogous to SIPs, many-to-many relations are not considered.

Semantic relationships [49] are a collection of five patterns capturing elementary types of coordination constraints among processes. As already described in the context of PHILharmonicFlows in subsection 4.4.1, semantic relationships incorporate the support for one-to-many and many-to-many process relations. Semantic relationships allow for concurrent and asynchronous process execution while affecting process execution only if required. By configuring basic semantic relationships, more complex coordination constraints may be represented. Semantic relationships are paradigm-independent as processes are abstracted with state-based-views, thus allowing for the coordination of processes modelled in any paradigm.

SIP and Correlation Patterns, both describe interactions that take into account relationships among interacting process instances. However, both pattern catalogues are limited to the activity-centric paradigm as interactions rely on message exchanges. Furthermore, many-to-many relations are not considered. In contrast, semantic relationships may be

used to model process interactions in many-to-many relations, independently from a specific paradigm.

In the literature, different modelling approaches with a main focus on process interactions have been proposed. Yet, most of them have not been implemented into a prototype that supports the design of process models. In the following, a small overview of those approaches is presented.

Proclets [53, 54] are lightweight processes which interact with one another via messages called performatives. Proclets allow setting cardinality restrictions for a message multicast, i.e. specifying the number of Proclets receiving a performative. Cardinality restrictions are fixed at design-time, though the exact recipients of a performative at run-time is mostly unknown at design-time. Proclets take asynchronous and concurrent execution into account. However, many-to-many relations between Proclets are not considered. Proclets are defined using an extended version of Petri nets, which supports the sending and receiving of performatives. A tool support for modelling process interactions based on the Proclets framework is not available so far.

Business Artifacts [23, 24] are business objects that consist of a lifecycle model and an information model. For process modelling the Guard-Stage-Milestone (GSM) meta-model [22] is used. The information model describes the data of a business object during its lifecycle while the lifecycle model describes the creation and evolution of the business object as it passes through a business. Relations between Business Artifacts play a role for the specification of the overall business process, however relation types are not explicitly defined. Business Artifacts interact with one another based on a sophisticated expression framework that is integrated into the process model. Cardinality restrictions can be configured with the expression framework. A prototype engine that supports the GSM meta-model has been developed by the IBM Research-Team for internal business operations.

The COREPRO approach [30, 31] focuses on the coordination of data-driven process structures in the engineering domain. Product components are modelled as data objects with a corresponding object lifecycle composed of states. Relations between data objects of the type one-to-many are explicitly considered. However, neither many-to-many relations nor cardinality restrictions are supported. Process interactions occur by coordinating the state of an object's lifecycle with the state of another object's lifecycle. Similar to PHILharmonicFlows, a data model and a lifecycle coordination model are defined and displayed separately. Major parts of the COREPRO modelling concept have been implemented in a prototype, the COREPRO Modeler [32], for conducting case studies in the automotive industry.

The BPMN standard [35] comprises choreography diagrams which explicitly focus on process interactions. Choreographies represent a type of business contract between two or more organisations. Choreography diagrams concentrate on the message flow between process participants instead of the individual tasks performed by each participant. Choreography diagrams posses few modelling elements, such that only the interactions

between senders and receivers are illustrated. Relation types between interacting processes can be deduced based on single-instance and multi-instance markers. However, analogous to collaboration diagrams, the support in restricting process cardinality is very limited. Several commercial modelling tools are offered for the design of choreography diagrams.

6
Summary and Outlook

In this chapter the main contributions of this thesis are briefly summarised. Furthermore, possible research directions for future work are presented.

6.1. Summary

This thesis aimed at providing a pattern catalogue describing the different types of interactions that may exist among process instances that have relationships of 1:n, m:1 and m:n. It was specific aim to capture patterns that go beyond the common 1:1 relationships. Furthermore, it was of fundamental importance to create a collection of patterns that explicitly capture process interactions in a paradigm-independent manner. For these, Process Coordination Patterns have been introduced. The current PCP catalogue, presented in Chapter 3, consists of seven patterns.

PCPs abstract process interactions as generally as possible; they try to capture the essence of the interaction, aiming to support a wide variety of process paradigms. Thereby, processes are considered at a comprehensive level as they are partitioned into different execution states that provide significant meaning for process interactions. PCPs take into account concurrent and asynchronous process executions. Interactions are based on execution dependencies among processes that are interrelated through different types of relationships. In the current PCP catalogue each pattern comprises variants for 1:1, 1:n / m:1 and m:n relationships. PCPs are a collection of elementary interactions derived from literature involving complex relationships. It is likely that through future research in this field, more rare patterns may be found and added to the current catalogue.

The degree to which two modelling approaches support modelling interactions captured by the PCPs has been evaluated in Chapter 4. From the two modelling approaches assessed for PCP support, one is based on the activity-centric paradigm, BPMN, and the other, PHILharmonicFlows, on a data-centric paradigm. For the evaluation, two scenarios exhibiting multiple process interactions, in particular one-to-many and many-to-many relationships, were selected. The evaluation results showed that PHILharmonicFlows as compared to BPMN provides a higher degree of support with respect to the capturing of

multiple process interactions. This follows from the fact that, PHILharmonicFlows has been designed with multiple process interactions in mind.

The evaluation performed in this thesis provided an initial insight of the scope and suitability of modelling approaches based on different paradigms. In future evaluations on PCP support, more modelling approaches could be compared and more precise conclusions about their similarities and differences with respect to the capturing of multiple process interactions could be drawn. Based on the design goal of supporting PCPs, existing interaction-centric modelling approaches could be modified. Furthermore, PCPs could serve as guideline for the design of new interaction-centric modelling approaches, ensuring that these support PCPs.

6.2. Outlook

To extend the theoretical base and practical use of PCPs, the following research topics have emerged as candidates for future work:

The Internet of Things (IoT) is a challenging topic in BPM, driving the need for new and different approaches to business process modelling and execution. The IoT represents a comprehensive environment that consists of a large number of smart devices, interconnecting heterogeneous physical objects to the internet. IoT poses a challenge in how to manage several single processes interacting with one another in both ways, synchronously and asynchronously, forming together one complex business process. PCPs could support process designers in tackling interacting heterogeneous processes in many-to-many relationship settings. A thorough empirical investigation of the PCPs framework for modelling large scale interactions shall demonstrate their applicability in practice.

The PCP framework intends enhancing a sufficiently detailed level of abstraction to serve as guidance for assessing the capabilities of modelling approaches based on different paradigms. However, this generalisation of processes leads to loss of detail, such that concrete modelling approaches do not necessarily conform to the overall definition of PCPs. The specific representation of state-based-views for different modelling approaches belonging to different paradigms is therefore a possible future research topic.

PCPs have been illustrated with simple graphic elements and constructs so far. For examining the use of PCPs in a series of PrMS as a means of assessing the specific capabilities of individual offerings for the support of multiple process interactions, a formal representation of PCPs is needed. A major focus of future work could therefore be the development of formal semantics for PCPs.

The current PCP catalogue has been developed based on current literature with focus on multiple process interactions. However, there is not much literature with focus on this topic. Therefore, using the PCP catalogue for the purpose of process language analysis

in the future, would be useful for obtaining a broader database such that potentially new PCPs could be discovered.

A

Appendix

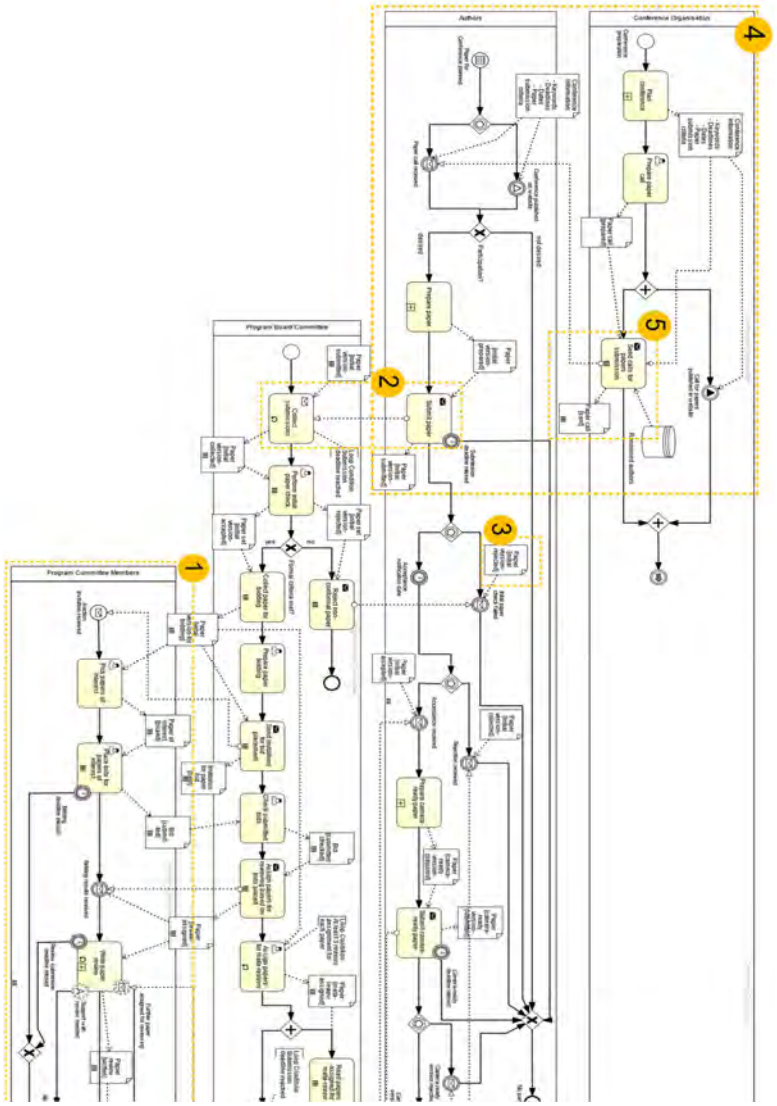


Figure A.1.: Paper Selection Process of a Conference - BPMN Model Part 1

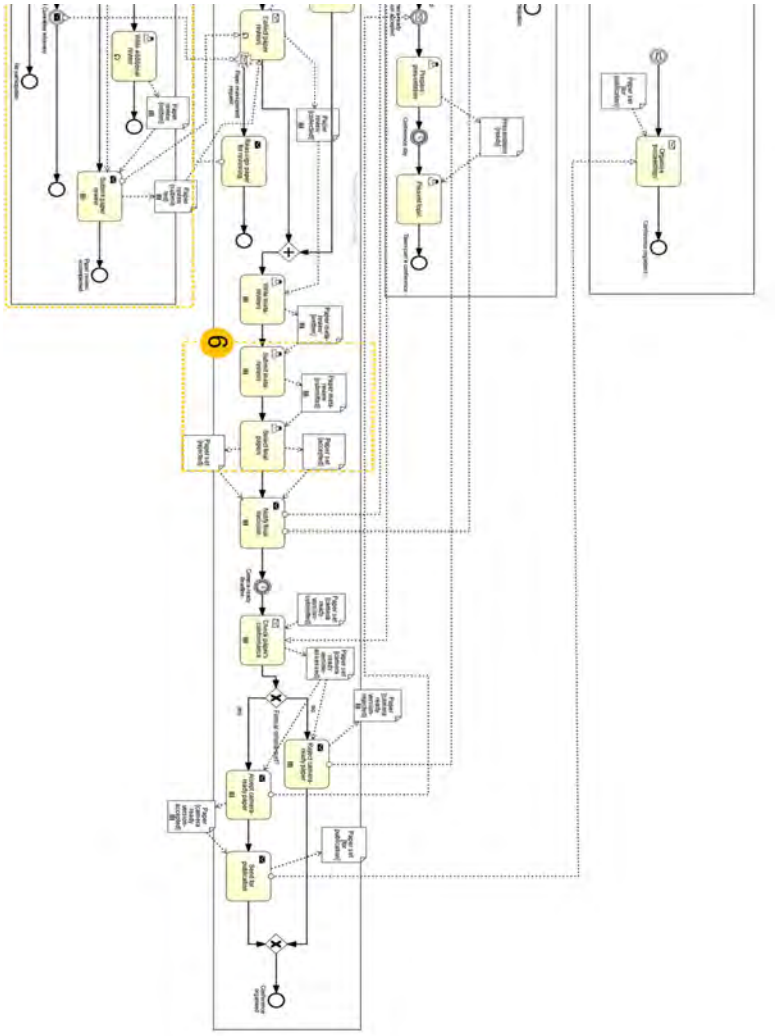


Figure A.1.: Paper Selection Process of a Conference - BPMN Model Part 2

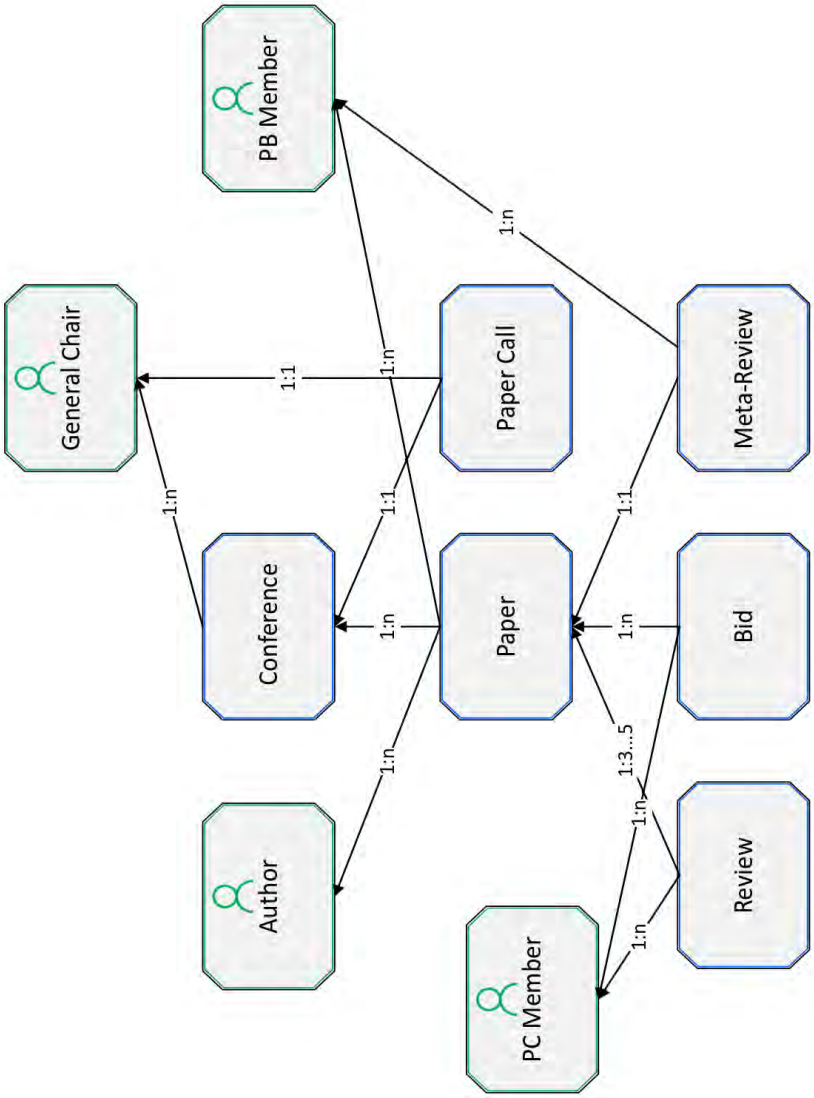


Figure A.2.: Paper Selection Process of a Conference - PHILharmonicFlows Data Model

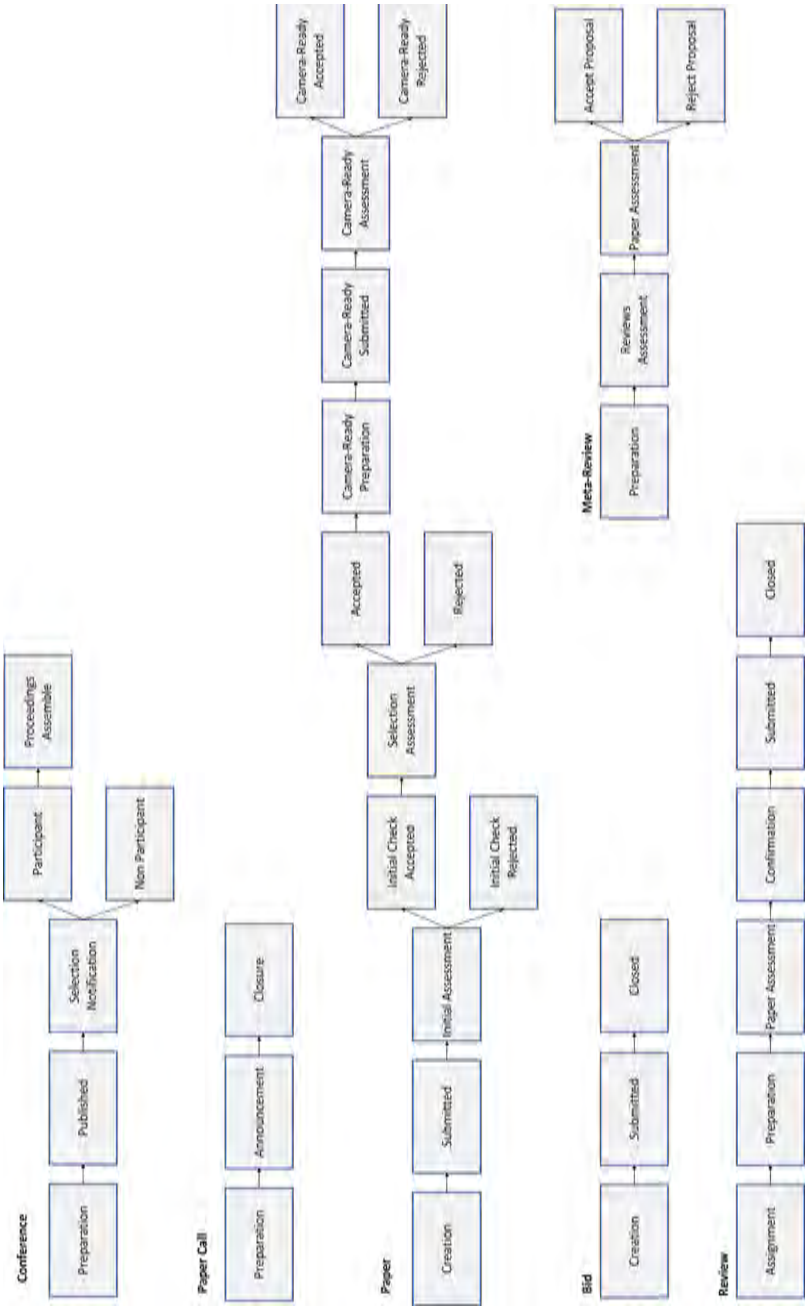


Figure A.3.: Paper Selection Process of a Conference - PHILharmonicFlows Lifecycle Processes

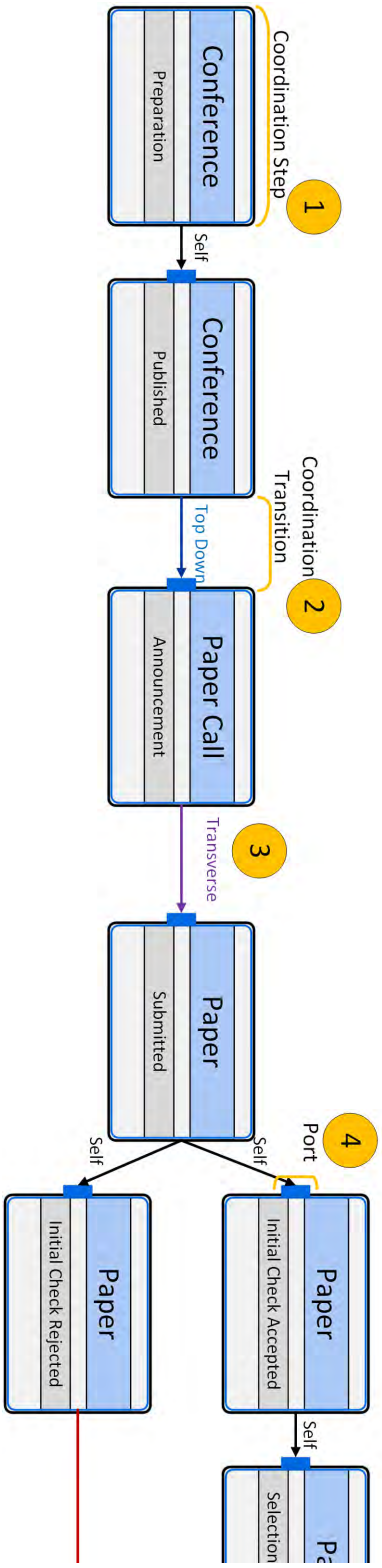


Figure A.4.: Paper Selection Process of a Conference - PHILharmonicFlows Coordination Process *Conference* Part 1

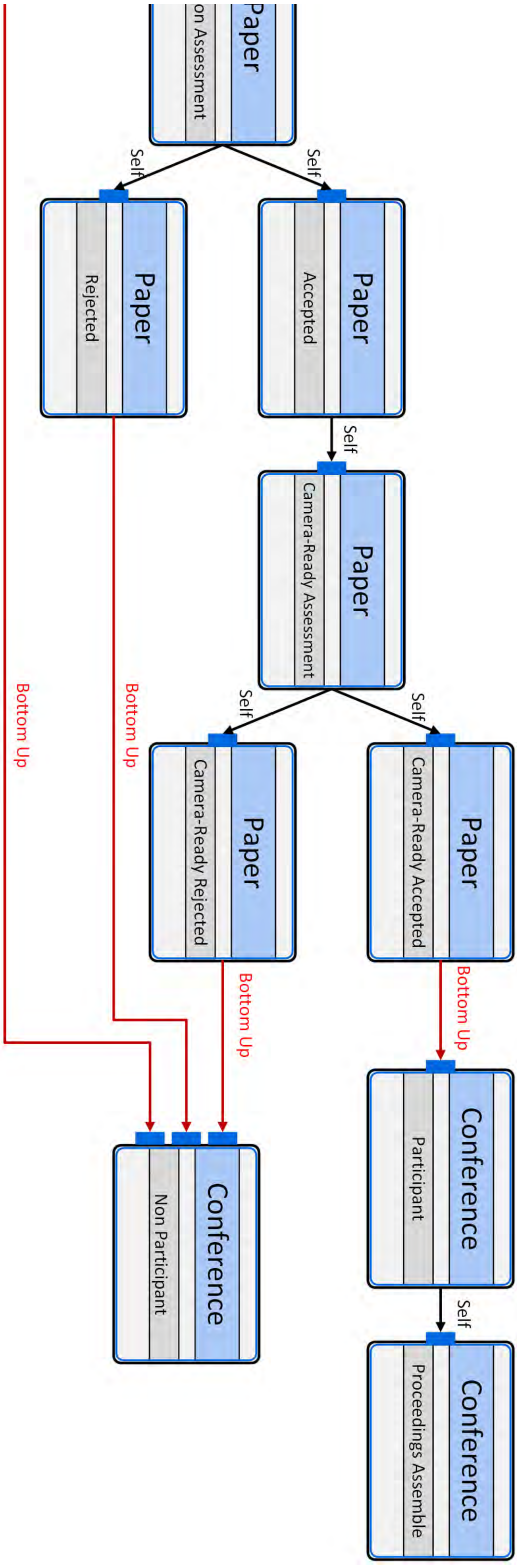


Figure A.4.: Paper Selection Process of a Conference - PHILharmonicFlows Coordination Process *Conference* Part 2

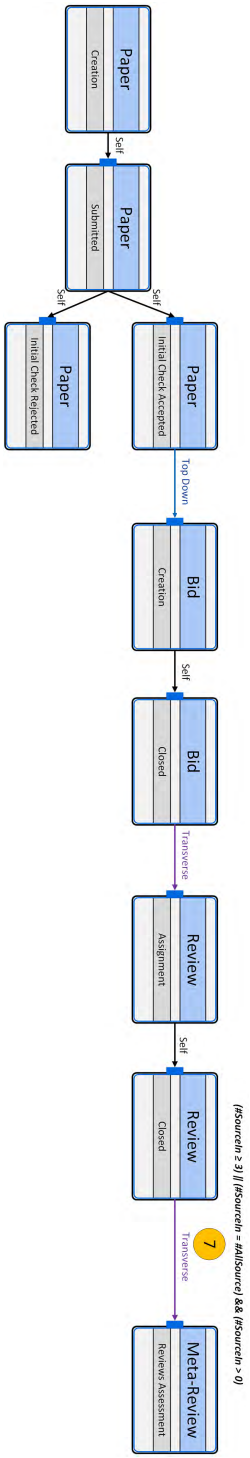


Figure A.5.: Paper Selection Process of a Conference - PHILharmonicFlows Coordination Process *Paper* Part 1

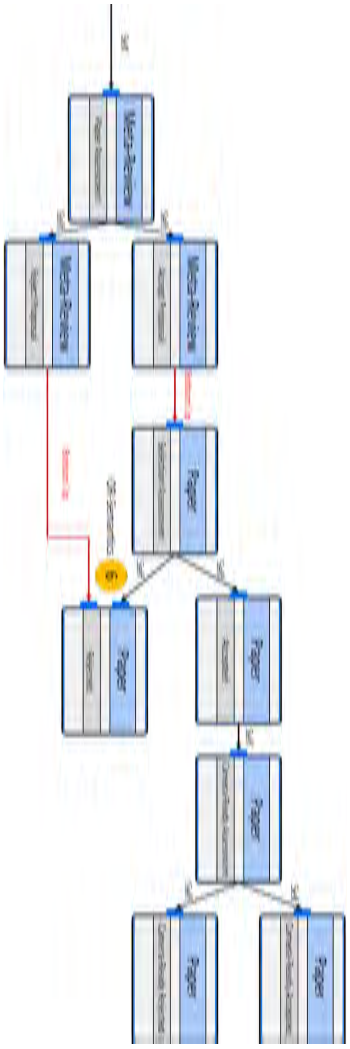


Figure A.5.: Paper Selection Process of a Conference - PHILharmonicFlows Coordination Process *Paper* Part 2

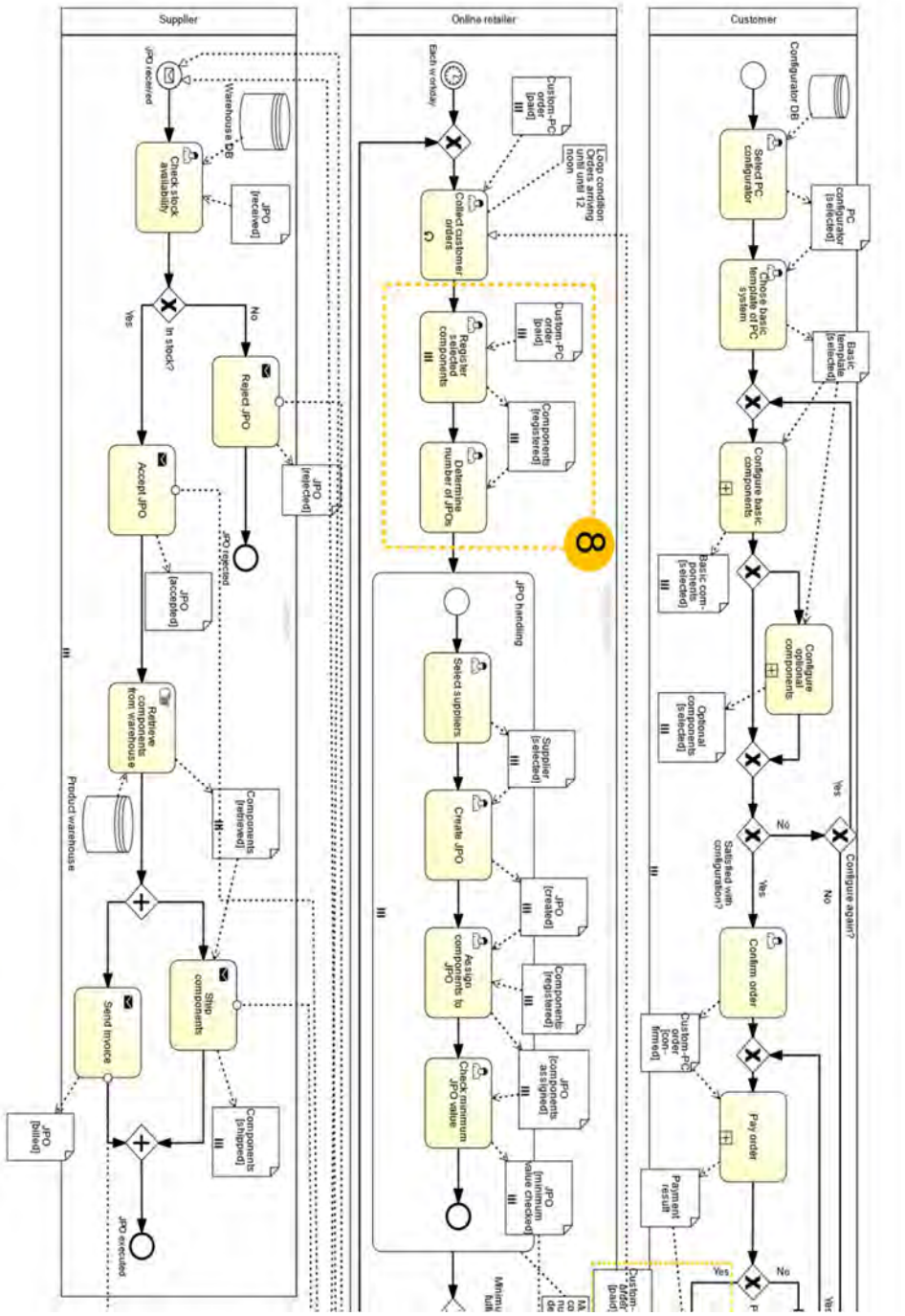


Figure A.6.: Built-to-Order Process - BPMN Model Part 1

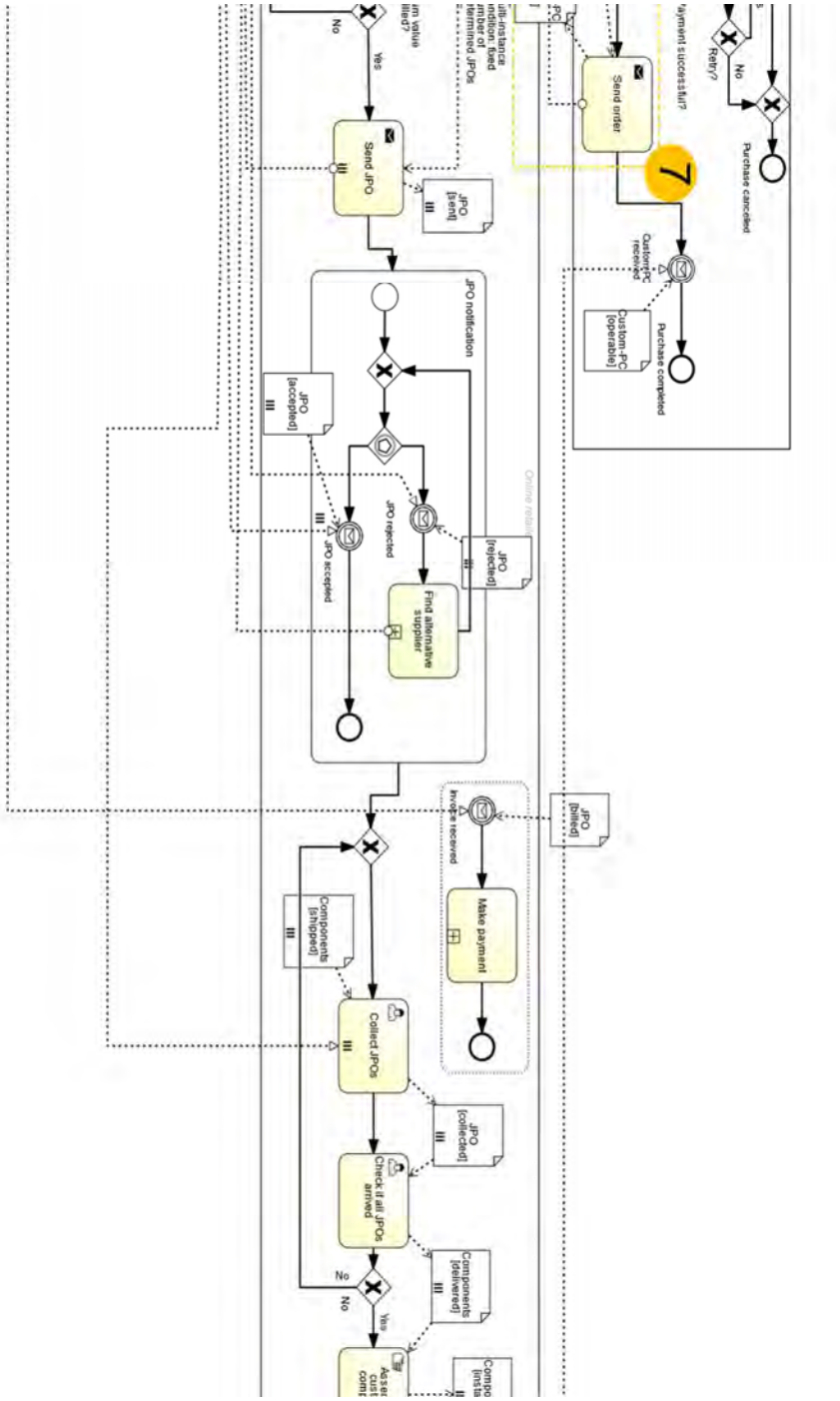


Figure A.6.: Built-to-Order Process - BPMN Model Part 2

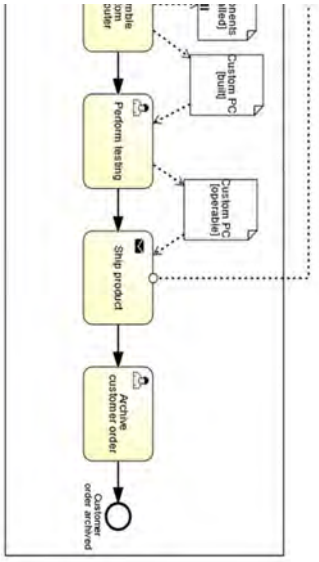


Figure A.6.: Built-to-Order Process - BPMN Model Part 3

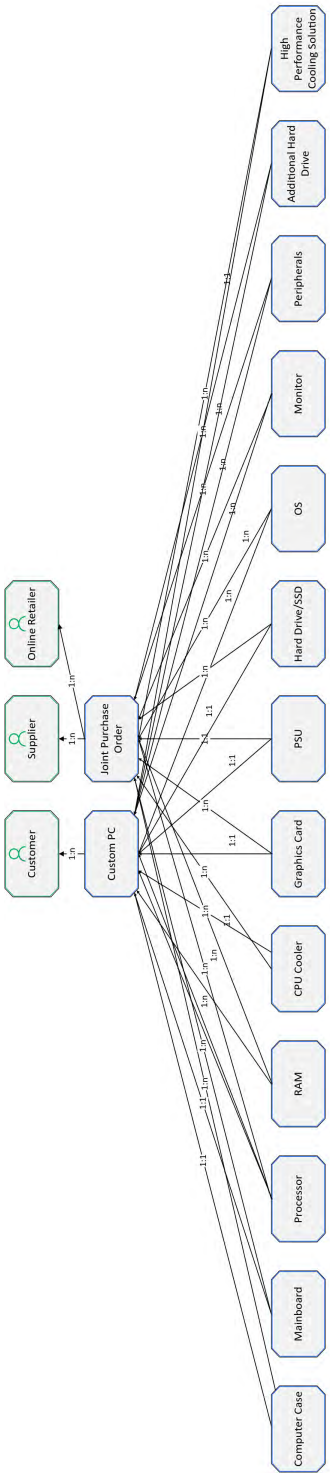


Figure A.7.: Built-to-Order Process - PHILharmonicFlows Data Model

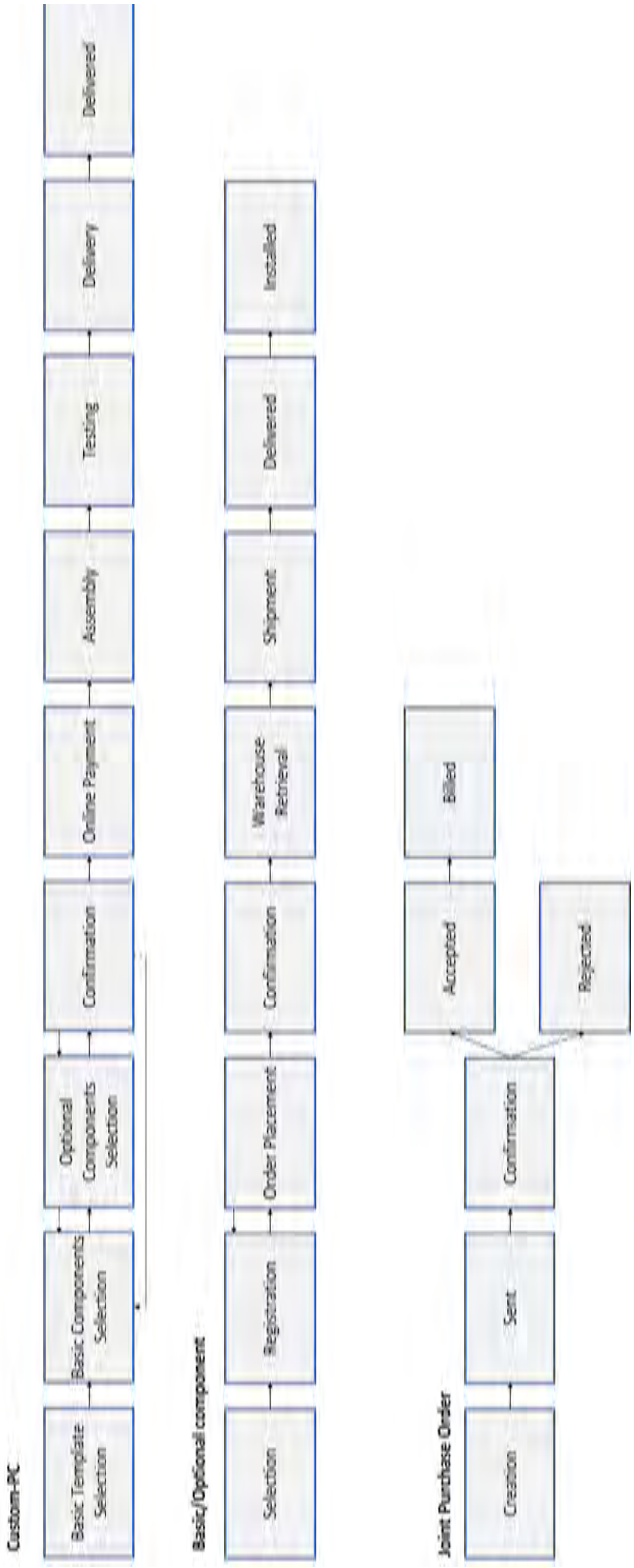


Figure A.8.: Built-to-Order Process - PHILharmonicFlows Lifecycle Processes

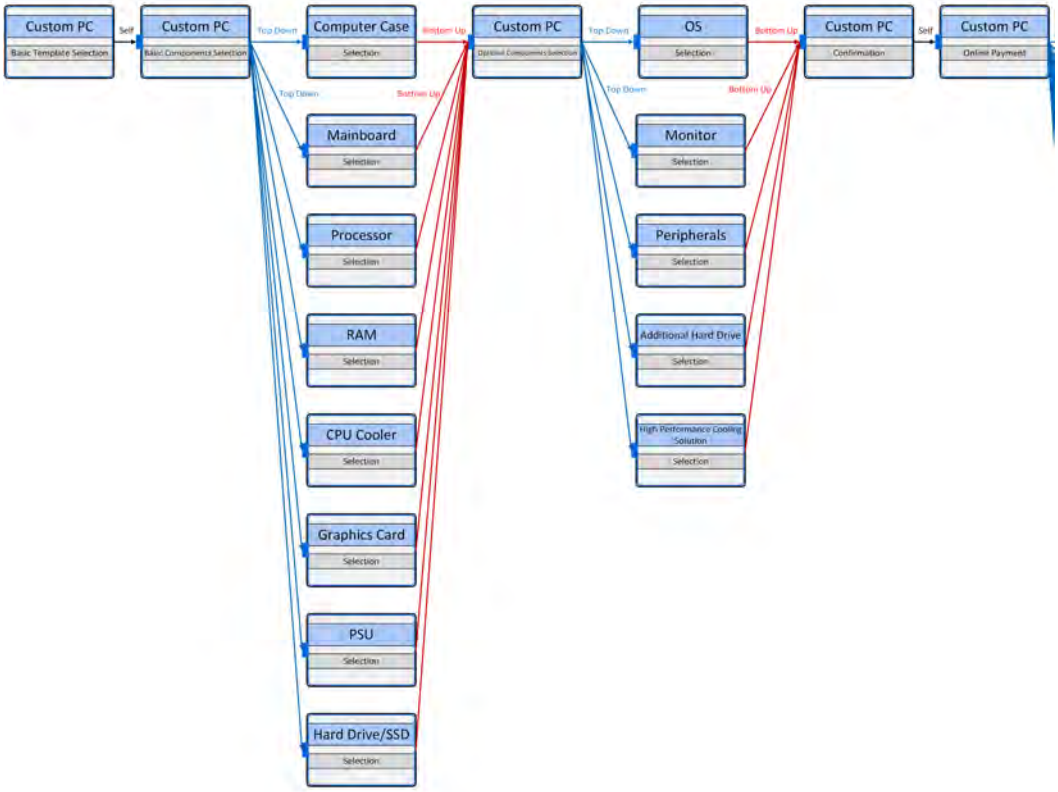


Figure A.9.: Built-to-Order Process - PHILharmonicFlows Coordination Process *Custom-PC* Part 1

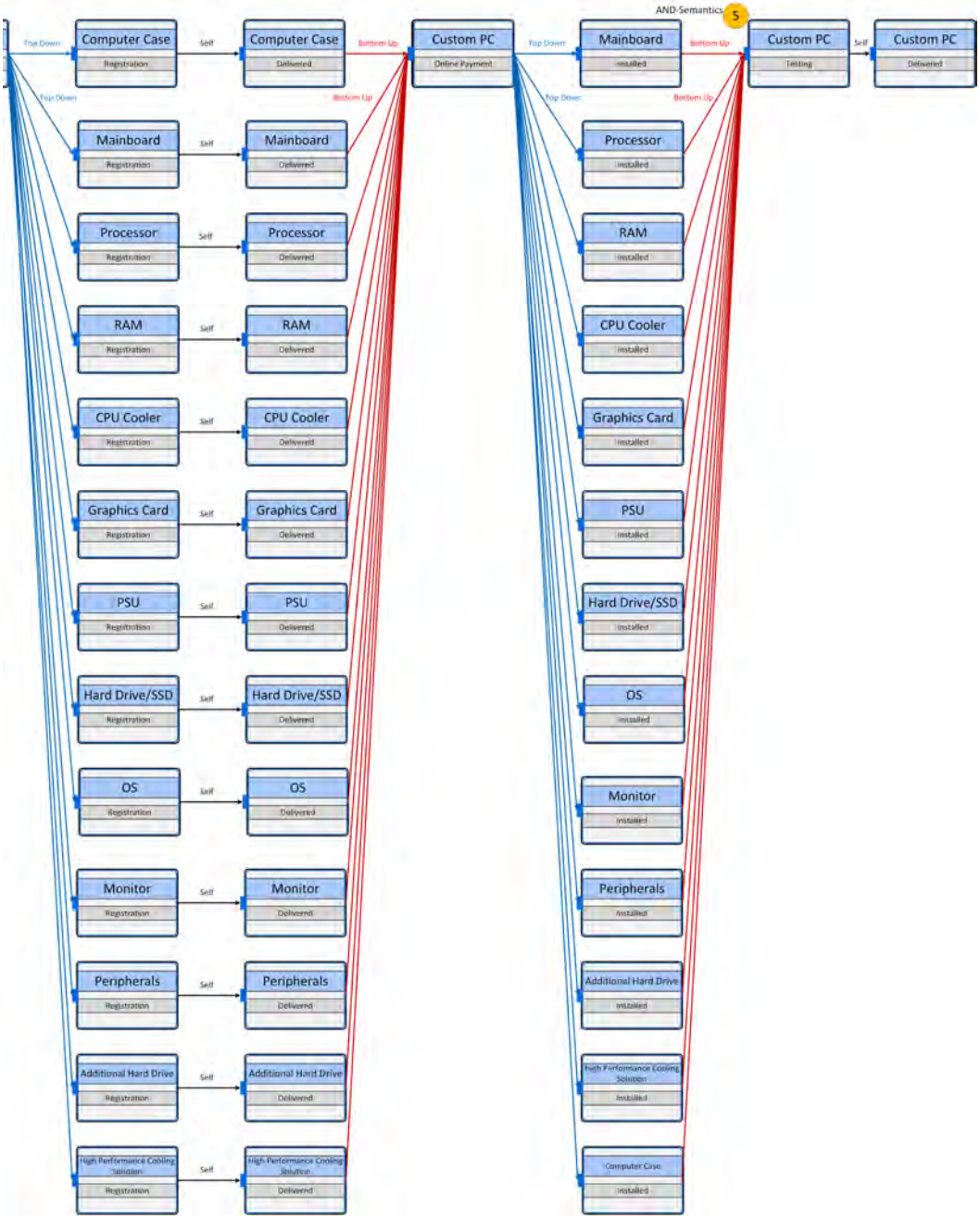


Figure A.9.: Built-to-Order Process - PHILharmonicFlows Coordination Process *Custom-PC* Part 2

Figure A.10.: Built-to-Order Process - PHILharmonicFlows Coordination Process *Joint Purchase Order* Part 1

Figure A.10.: Built-to-Order Process - PHILharmonicFlows Coordination Process *Joint Purchase Order* Part 2

List of Tables

List of Figures

Bibliography

[1] Andrews, K., Steinau, S., Reichert, M.: Enabling Fine-Grained Access Control in Flexible Distributed Object-Aware Process Management Systems. In: 2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC). pp. 143–152 (2017)

[2] Awad, A., Decker, G., Lohmann, N.: Diagnosing and Repairing Data Anomalies in Process Models. In: Business Process Management Workshops. pp. 5–16. Springer Berlin Heidelberg (2010)

[3] Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation Patterns in Service-Oriented Architectures. In: Fundamental Approaches to Software Engineering. pp. 245–259. Springer Berlin Heidelberg (2007)

[4] Barros, A., Dumas, M., ter Hofstede, A.: Service Interaction Patterns. In: Business Process Management. pp. 302–318. Springer Berlin Heidelberg (2005)

[5] Buhl, H.U., Röglinger, M., Stöckl, S., Braunwarth, K.S.: Value Orientation in Process Management. Business & Information Systems Engineering **3**(3), 163–172 (2011)

[6] Cohn, D., Hull, R.: Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering **32**(3) (2009)

[7] Corradini, F., Muzi, C., Re, B., Rossi, L., Tiezzi, F.: Animating Multiple Instances in BPMN Collaborations: From Formal Semantics to Tool Support. In: Business Process Management. pp. 83–101. Springer International Publishing (2018)

[8] Decker, G., Barros, A.: Interaction Modeling Using BPMN. In: Business Process Management Workshops. pp. 208–219. Springer Berlin Heidelberg (2008)

[9] Decker, G., Dijkman, R., Dumas, M., García-Bañuelos, L.: Transforming BPMN Diagrams into YAWL Nets. In: Business Process Management. pp. 386–389. Springer Berlin Heidelberg (2008)

[10] Decker, G., Kopp, O., Leymann, F., Pfitzner, K., Weske, M.: Modeling Service Choreographies Using BPMN and BPEL4Chor. In: Advanced Information Systems Engineering. pp. 79–93. Springer Berlin Heidelberg (2008)

[11] Derrick, J., Boiten, E.A.: Refinement in Z and object-Z: Foundations and Advanced Applications. Springer Science & Business Media (2013)

[12] Dijkman, R., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information and Software Technology **50**(12), 1281–1294 (2008)

[13] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer Berlin Heidelberg (2018)

[14] Dumas, M., van der Aalst, W., ter Hofstede, A.: Process-aware Information Systems: Bridging People and Software through Process Technology. John Wiley & Sons (2005)

[15] Eder, J., Lehmann, M.: Synchronizing Copies of External Data in Workflow Management Systems. In: Advanced Information Systems Engineering. pp. 248–261. Springer Berlin Heidelberg (2005)

[16] Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.: Many-to-Many: Some Observations on Interactions in Artifact Choreographies. In: Proceedings of the 3rd Central-European Workshop (ZEUS). vol. 705, pp. 9–15. CEUR-WS. org, CEUR Workshop Proceedings (2011)

[17] Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST Standard for Role-based Access Control. ACM Transactions on Information and System Security **4**(3), 224–274 (2001)

[18] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Abstraction and Reuse of Object-Oriented Design. In: ECOOP' 93 — Object-Oriented Programming. pp. 406–431. Springer Berlin Heidelberg (1993)

[19] de Giacomo, G., Dumas, M., Maggi, F.M., Montali, M.: Declarative Process Modeling in BPMN. In: Advanced Information Systems Engineering. pp. 84–100. Springer International Publishing (2015)

[20] Günther, C., van der Aalst, W.: Process Mining in Case Handling Systems. In: Multikonferenz Wirtschaftsinformatik 2006 (MKWI). pp. 125–137. GITO-Verlag Berlin (2006)

[21] Houy, C., Fettke, P., Loos, P., van der Aalst, W., Krogstie, J.: BPM-in-the-Large – Towards a Higher Level of Abstraction in Business Process Management. In: E-Government, E-Services and Global Processes. pp. 233–244. Springer Berlin Heidelberg (2010)

[22] Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles. In: Web Services and Formal Methods. pp. 1–24. Springer Berlin Heidelberg (2011)

[23] Hull, R., Damaggio, E., de Masellis, R., Fournier, F., Gupta, M., Heath, F., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculin, R.: Business Artifacts with Guard-stage-milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In: Proceedings of the 5th ACM International Conference on Distributed Event-based System. pp. 51–62. ACM (2011)

[24] Hull, R., Narendra, N.C., Nigam, A.: Facilitating Workflow Interoperation Using Artifact-Centric Hubs. In: Service-Oriented Computing. pp. 1–18. Springer Berlin Heidelberg (2009)

[25] 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)

[26] Lanz, A., Weber, B., Reichert, M.: Time Patterns for Process-aware Information Systems. Requirements Engineering **19**(2), 113–141 (2014)

[27] Meidan, A., García-García, J.A., Escalona, M.J., Ramos, I.: A Survey on Business Processes Management Suites. Computer Standards & Interfaces **51**, 71–86 (2017)

[28] Meyer, A., Pufahl, L., Fahland, D., Weske, M.: Modeling and Enacting Complex Data Dependencies in Business Processes. In: Business Process Management. pp. 171–186. Springer Berlin Heidelberg (2013)

[29] Meyer, A., Weske, M.: Activity-Centric and Artifact-Centric Process Model Roundtrip. In: Business Process Management Workshops. pp. 167–181. Springer International Publishing (2014)

[30] Müller, D., Reichert, M., Herbst, J.: Data-driven Modeling and Coordination of Large Process Structures. In: On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS. pp. 131–149. Springer Berlin Heidelberg (2007)

[31] Müller, D., Reichert, M., Herbst, J.: A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures. In: Advanced Information Systems Engineering. pp. 48–63. Springer Berlin Heidelberg (2008)

[32] Müller, D., Reichert, M., Herbst, J., Poppa, F.: Data-driven Design of Engineering Processes with COREPROModeler. In: 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). pp. 376–378 (2007)

[33] Mulyar, N.A.: Patterns for Process-aware Information Systems: An Approach based on Colored Petri Nets. Ph.D. thesis, Technische Universiteit Eindhoven (2009)

[34] Nigam, A., Caswell, N.S.: Business artifacts: An Approach to Operational Specification. IBM Systems Journal **42**(3), 428–445 (2003)

[35] Object Management Group: Business Process Model and Notation (BPMN), Version 2.0 (2011), `http://www.omg.org/spec/BPMN/2.0`

[36] Popova, V., Fahland, D., Dumas, M.: Artifact Lifecycle Discovery. International Journal of Cooperative Information Systems **24**(1) (2015)

[37] Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer Berlin Heidelberg (2012)

[38] Reijers, H.A., Rigter, J.H., van der Aalst, W.: The Case Handling Case. International Journal of Cooperative Information Systems **12**(3), 365–391 (2003)

[39] Reijers, H.A., Vanderfeesten, I., Plomp, M.G.A., van Gorp, P., Fahland, D., van der Crommert, W., Garcia, H.D.D.: Evaluating Data-centric Process Approaches: Does the Human Factor factor in? Software & Systems Modeling **16**(3), 649–662 (2017)

[40] Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall PTR (1997)

[41] von Rosing, M., von Scheel, H., Scheer, A.W.: The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM. Morgan Kaufmann Publishers Inc (2014)

[42] Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W.: Workflow Data Patterns: Identification, Representation and Tool Support. In: Conceptual Modeling – ER 2005. pp. 353–368. Springer Berlin Heidelberg (2005)

[43] Russell, N., van der Aalst, W., ter Hofstede, A.: Workflow Patterns: The Definitive Guide. The MIT Press (2016)

[44] Sadiq, S., Orlowska, M., Sadiq, W., Foulger, C.: Data Flow and Validation in Workflow Modelling. In: Proceedings of the 15th Australasian Database Conference. pp. 207–214 (2004)

[45] von Stackelberg, S., Putze, S., Mülle, J., Böhm, K.: Detecting Data-Flow Errors in BPMN 2.0. Open Journal of Information Systems **1**(2), 1–19 (2014)

[46] Steinau, S., Andrews, K., Reichert, M.: Flexible Data Acquisition in Object-aware Process Management. In: 7th Int'l Symposium on Data-driven Process Discovery and Analysis (SIMPDA). pp. 113–127. CEUR-WS.org (2017)

[47] Steinau, S., Andrews, K., Reichert, M.: Modeling Process Interactions with Coordination Processes. In: On the Move to Meaningful Internet Systems. OTM 2018 Conferences. pp. 21–39. Springer International Publishing (2018)

[48] Steinau, S., Andrews, K., Reichert, M.: The Relational Process Structure. In: Advanced Information Systems Engineering. pp. 53–67. Springer International Publishing (2018)

[49] Steinau, S., Künzle, V., Andrews, K., Reichert, M.: Coordinating Business Processes Using Semantic Relationships. In: 2017 IEEE 19th Conference on Business Informatics (CBI). pp. 33–42. IEEE Computer Society Press (2017)

[50] Steinau, S., Marrella, A., Andrews, K., Leotta, F., Mecella, M., Reichert, M.: DALEC: a Framework for the Systematic Evaluation of Data-centric Approaches to Process Management Software. Software & Systems Modeling (2019)

[51] Sun, Y., Xu, W., Su, J.: Declarative Choreographies for Artifacts. In: Service-Oriented Computing. pp. 420–434. Springer Berlin Heidelberg (2012)

[52] Tanenbaum, A.S., Bos, H.: Modern Operating Systems. Pearson Education (2014)

[53] van der Aalst, W., Barthelmess, P., Ellis, C.A., Wainer, J.: Workflow Modeling using Proclets. In: Cooperative Information Systems. pp. 198–209. Springer Berlin Heidelberg (2000)

[54] van der Aalst, W., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A Framework for Lightweight Interacting Workflow Processes. International Journal of Cooperative Information Systems **10**(4), 443–481 (2001)

[55] van der Aalst, W., Mooij, A.J., Stahl, C., Wolf, K.: Service Interaction: Patterns, Formalization, and Analysis. In: Formal Methods for Web Services: 9th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, pp. 42–88. Springer Berlin Heidelberg (2009)

[56] van der Aalst, W., Stoffele, M., Wamelink, J.W.: Case Handling in Construction. Automation in Construction **12**(3), 303–320 (2003)

[57] van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases **14**(1), 5–51 (2003)

[58] van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: A New Paradigm for Business Process Support. Data & Knowledge Engineering **53**(2), 129–162 (2005)

[59] Weber, B., Reichert, M., Rinderle-Ma, S.: Change Patterns and Change Support Features–Enhancing Flexibility in Process-aware Information Systems. Data & Knowledge Engineering **66**(3), 438–466 (2008)

[60] Weske, M.: Business Process Management Architectures. Springer (2012)

[61] Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A., Russell, N.: On the Suitability of BPMN for Business Process Modelling. In: Business Process Management. pp. 161–176. Springer Berlin Heidelberg (2006)

[62] Wong, P.Y.H., Gibbons, J.: A Process Semantics for BPMN. In: Formal Methods and Software Engineering. pp. 355–374. Springer Berlin Heidelberg (2008)

Name: Marisol Schwarz Rosado Matrikelnummer: 947052

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Marisol Schwarz Rosado