Comparing Imperative and Declarative Process Models

Michaela Baumann

Institute of Computer Science
University of Bayreuth, Germany
michaela.baumann@uni-bayreuth.de

Abstract. The field of process model similarity matching is well examined for imperative process models like BPMN models, Petri nets, or EPCs where a lot of different measuring techniques exist. For the recently upcoming declarative process models, generally providing more flexibility than imperative models, however, there is a lack of comparison methods. Along with their advantage of providing more flexibility, declarative process models have a disadvantage in comprehending the models, especially the models' behavior. To overcome this problem, a comparison of imperative and declarative models is reasonable to check whether the declarative model represents a desired behavior which is easier to express and validate in an imperative notation. The work at hand provides a method based on flow dependencies, abstracting from the modeling type, for comparing two process models. It uses not only information about control-flow, but also data-based dependencies between process activities.

Keywords: Process Model Comparison, Process Model Similarity, Behavioral Similarity, Behavioral Profile, Flow Dependency

1 Introduction and Discussion of Related Work

Deriving similarity between process models has a lot of use cases mentioned in literature, cf. [1], [8], [14]. On the one hand, it is often referred to in the context of large model repositories and their management. When a repository grows, the identification of duplicate models [13] and model variants [15], [24], or the reuse of model parts [22] becomes a challenging issue. On the other hand, also the comparison of only a small set of models with reference models [21] for verifying compliance [7] is one application field of measuring similarity of process models. The method is thereby usually split into two steps: in the first step, a mapping is established between the two models that shall be compared. In a second step, a similarity value is computed based on the mapping of step one. This procedure can also be iterated, depending on the underlying algorithm, to find the best mapping that provides the highest similarity value. The similarity value can include several aspects, referred to as the five perspectives [16] of a process model: task description, control-flow, data-flow, as well as human and non-human resources. For process models representable as graphs, also graph-based comparison techniques can be applied. In earlier work, the main focus lies on label similarity (semantic and syntactic similarity), contextual similarity, structural similarity like graph-edit distance, and behavioral, i.e., control-flow similarity [5], [9], [10], [12]. In more recent work, also other process perspectives are taken into account: (non-)human resources [6] and data, more precisely data-flow [2]. In [3], a method for combining all perspective similarities is given.

What all of these approaches have in common is that they are designed for imperative process models, i.e., models used for designing routine processes. Some methods are explicitly suitable for petri nets or event process chains, like [1], [9], for dependency graphs [2], or for abstracted, graph-like process models, e.g., [3], [17]. For declarative process models, i.e., models for designing agile processes, however, research concerning similarity matching is not highly developed. Similarity matching for declarative models is suggested in [27] via a transformation of declarative models to a finite state automaton representation and applying known matching techniques on the automaton. This approach is restricted to models representable as finite state automata. Other approaches are mentioned in [4] like using simulation of execution traces and conformity checking, applying execution patterns, or directly comparing the constraints of a declarative process model on the logical level. The approaches are, however, not fully fomalized. Arbitrary comparison of both imperative and declarative models is, to the best of the author's knowledge, not yet investigated although this is a promising field of research. Declarative models provide a good level of flexibility [20] but also impose a difficulty to fully understand the models [29]. In order to better understand a declarative process model, a comparison with an imperative model designated to represent (parts of) the declarative model can be carried out. The work at hand presents one method for comparing two models with respect to their behavior. The approach is able to give hints that one model is a subsumption of the other model but also points out the differences. It makes use of the dependencies between every two model elements, which we call *flow dependencies*. This approach is similar to the behavioral profile techniques proposed, for example, in [2], [18], [25], [28], adapting and enlarging them for the application with declarative process models and incorporating not only basic control-flow information but also data-flow.

The structure of the paper is as follows: in Section 2, imperative and declarative process models are defined to provide a consistent basis. Section 3 describes how we can extract flow dependencies involving two activities from both imperative and declarative process models, catching as much information as possible available in both model types. With help of a hierarchy of the dependencies, Section 4 illustrates how two process models can be compared. This allows for the proposition whether the models are in a subsumption relation, contradictory, or not comparable. If there are inconsistencies, they can be located. Section 5 concludes the paper with a short discussion about current limitations and an outlook for future work, involving the derivation of a similarity measure.

2 Imperative and Declarative Process Models

In order to abstract from a specific imperative process modeling language, we regard imperative process models as models of the form stated in Definition 1. Agents, for example lanes in BPMN, are not considered in this definition.

Definition 1 (Imperative Process Model). *A (graphical) imperative process model G is a tuple (N, E, τ, λ) where*

- *N is a set of nodes, $E \subseteq N \times N$ is a set of edges, $\tau : N \to \mathcal{T}$ is a function mapping nodes to types, and $\lambda : N \to \mathcal{L}$ is a function mapping nodes to labels.*
- *(N, E) is a connected graph.*
- *We define five different types of nodes: $\mathcal{T} = \{$start event, end event, activity, AND-gateway, XOR-gateway$\}$.*
  - *There is exactly one start event (no incoming edge, one outgoing edge) and one end event (one incoming edge, no outgoing edge).*
  - *Each activity has exactly one incoming and one outgoind edge and consists of three different parts: task description, attached incoming, and attached outgoing data. Function λ is a three-dimensional mapping that maps each node to a task description (λ_1), a data identifier for incoming data (λ_2), and a data identifier for outgoing data (λ_3).*
  - *Each gateway is either a split or a merge gateway. Split gateways have exactly one incoming edge and at least two outgoing edges. Merge gateways have at least two incoming and exactly one outgoing edge.*

Definition 1 allows for process models with loops. Loops are usually modeled with XOR-gateways (not AND-gateways) where one of the outgoing edges points backward to a merge XOR-gateway. Fig. 1 shows an imperative process model in BPMN with activities A to H, XOR- and AND-gateways, and a data object flowing from E to F. More precisely, the data connection according to Definition 1 is $\lambda_2(F) = \delta$ and $\lambda_3(E) = \delta$ with δ being the data identifier. The example model is in accordance with Definition 1.

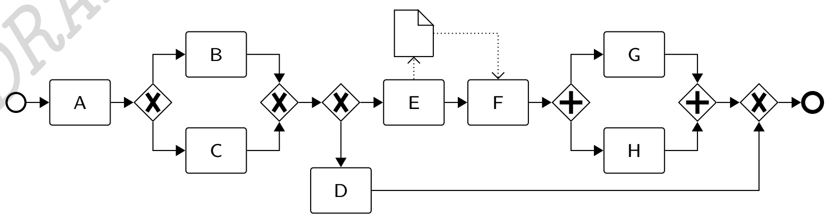

Fig. 1. Example of an imperative process model (BPMN model).

Declarative process models are of the form defined in Definition 2. This definition is in accordance with [19].

Definition 2 (Declarative Process Model). *A declarative process model D consists of two basic elements: activities $\mathcal{A}$ and rules $\mathcal{R}$. Both sets are finite. Activities have at least two events: start and complete. Furthermore, there is a set of data, resp. data identifiers. The rules are of the form $Body \Rightarrow Head$. Both $Body$ and $Head$ represent logical expressions over process events (start and completion of activities, writing of data, etc.) and process variables and can include a temporal ordering. Each rule either connects two activities (control-flow rules, $\mathcal{R}_c \subseteq \mathcal{R}$), an activity and a data identifier or two data identifiers (data rules, $\mathcal{R}_d \subseteq \mathcal{R}$).*

Edges as well as functions λ and τ are not explicitly given for declarative process models. Edges and λ are given implicitly within the rules. Different types of nodes do not exist. Note that the expression $Body \Rightarrow Head$ can also be written as $\neg(Body \wedge \neg Head)$ or $\neg Body \vee Head$ and that $\{Body \Rightarrow Head\} = \{\neg Head \Rightarrow \neg Body\}$. With this it is possible to formulate all rules with a positive *Body*. The restriction that each rule is able to connect only two objects is important to get reasonable comparison results between an imperative and a declarative model. An exemplary declarative process model is given in Table 1 and a list of all possible rules enabled through Definition 2 is provided in Table 2 regarding two events: start and complete. The example process model of Table 1 consists of $\mathcal{A} = \{$A, B, C, D, E, F, G, H$\}$ and rules $\mathcal{R} = \{1, \ldots, 8\}$. Variables t and s are points of time and δ denotes a specific data object. Rules 1, 2, 3, and 7 are each connecting two activites whereas 4, 5, 6, and 8 are connecting an activity with a data identifier. Rules 1 and 4 contain temporal ordering information. For the other rules, t and s are indepedent, i.e., they hold for $s \leq t$ and for $s > t$.

No.	Rule	Explanation
1	start of B at $t \Rightarrow$ complete of A at $s < t$	"A has to be completed before B can start"
2	complete of B at $t \Rightarrow$ not complete of C at s	"B can only be finished when C is not completed yet and C may not be completed once B is completed"
3	complete of C at $t \Rightarrow$ not complete of B at s	(see rule 2)
4	start of F at $t \Rightarrow$ write of δ at $s < t$	"F can only be started when data δ is already available, i.e., F consumes δ"
5	complete of E at $t \Rightarrow$ write of δ at s	"The completion of E requires that data δ is written, i.e., E causes the production of δ"
6	start of G at $t \Rightarrow$ write of δ at s	(see rule 5 with start instead of completion)
7	complete of H at $t \Rightarrow$ complete of G at s	"Whenever H is proceeded, G needs to be proceeded, too"
8	write of δ at $t \Rightarrow$ not complete of D at s	"As soon as δ is produced, D may not be completed"

Table 1. Example of a declarative process model with rule explanations.

control-flow rules	data rules
s/c of A at $t \Rightarrow$ (not) s/c of B at s	s/c of A at $t \Rightarrow$ (not) write of δ at s
s/c of A at $t \Rightarrow$ (not) s/c of B at $s < t$	s/c of A at $t \Rightarrow$ (not) write of δ at $s < t$
s/c of A at $t \Rightarrow$ (not) s/c of B at $s > t$	s/c of A at $t \Rightarrow$ (not) write of δ at $s > t$
	write of δ at $t \Rightarrow$ (not) s/c of B at s
	write of δ at $t \Rightarrow$ (not) s/c of B at $s < t$
	write of δ at $t \Rightarrow$ (not) s/c of B at $s > t$
	write of δ at $t \Rightarrow$ (not) write of ε at s
	write of δ at $t \Rightarrow$ (not) write of ε at $s < t$
	write of δ at $t \Rightarrow$ (not) write of ε at $s > t$

Table 2. List of all possible rules in a declarative process model with A and B denoting activities and δ and ε data identifiers; s/c stands for either start or complete; t and s are time variables.

3 Flow Dependencies in Process Models

We determine flow dependencies in process models only locally to not get into a potentially irresolvable problem when regarding all possible executions. Therefore, we call them (local) dependencies. Note that a dependency is not the same as an edge from Definition 1. Flow dependencies (or only "dependencies") can be extracted from process behavior/control-flow, but also from other process perspectives (see also [28] for an example). In the work at hand, we exemplarily show this for the data perspective.

Definition 3 (Dependency Type). *We distiguish between two different types of dependencies within a process model: control-flow-related dependencies (c) and data-related dependencies (d). Every dependency can either be a mandatory or an optional ordering dependency, a mutual-exclusion dependency (always symmetric), or a symmetric or non-symmetric existence dependency. For each pair of activities, both an existence dependency and an ordering dependency can be stated. We define the symbols for the dependencies as shown in Tab. 3.*

	control-flow	data
mandatory ordering	${}^{c}\twoheadrightarrow$	${}^{d}\twoheadrightarrow$
optional ordering	${}^{c}\rightarrow$	${}^{d}\rightarrow$
symmetric existence	${}^{c}==$	${}^{d}==$
non-symmetric existence	${}^{c}=>$	${}^{d}=>$
exclusive existence	${}^{c}><$	${}^{d}><$

Table 3. Possible dependency types within a process model.

Additionally to the dependencies defined above, if it is not possible to state an ordering for two activities which is, e.g., the case for the exclusive existence, we write the symbol $-$ (no ordering, absence of any order).

The order of two activities A and B is always stated in both directions: (A,B) and (B,A). For each direction, we have a pre- and a postconditional ordering. A mandatory postconditional ordering $\twoheadrightarrow$(A,B) means that after A, B has to be executed. A mandatory (preconditional) ordering $\twoheadleftarrow$(A,B) means that before B, A has to be executed. It is not possible to have both $\twoheadrightarrow$(A,B) and $\twoheadrightarrow$(B,A) as this would be an infinite cycle. An optional postconditional ordering $\rightarrow$(A,B) means that it is possible to do B after A but there are also ways to finish the process without doing B after A. An optional preconditional ordering $\leftarrow$(A,B) means that it is possible but not necessary to do A before B. Ordering arrows for one tuple of activities are combined, e.g., $\twoheadleftarrow\!\!\longrightarrow$(A,B). It is not possible to have ordering arrows with an arrowhead on only one side, e.g., to have only $\rightarrow$(A,B) but not $\leftarrow$(A,B) as this is simply a not possible behavior. Thus, for each activity tuple we have four different kinds of ordering dependencies: $\leftrightarrow$(A,B), $\longleftarrow\!\!\twoheadrightarrow$(A,B), $\twoheadleftarrow\!\!\longrightarrow$(A,B), or $-(A, B)$.

Non-symmetric existence =>(A,B) means that if A is done in a process instance then B needs to be done as well. The symmetric-existence dependency ==(A,B) means that if A is done in a process instance, B needs to be done as well, and the other way round (if =>(A,B) and =>(B,A), we have ==(A,B)). A mandatory ordering always implies an existence dependency ($\twoheadrightarrow$(A,B) implies =>(A,B), for example) but not the other way round. The exclusive existence dependency ><(A,B) means that as soon as A is executed, B can no longer be executed and the other way round. The distinction between ordering and existence dependencies can also be found in [11] where they are called *temporal* and *dependency relationships*.

3.1 Dependencies in Imperative Process Models

In the following, we show how to derive flow dependencies from an imperative process model. Dependencies in an imperative process model will only be defined for activities and neither for start and end event nor for gateways. To assign the correct dependencies to every pair of activities, we analyze the structure of the process model as done in [23]. There, a process model is decomposed into process fragments. The fragments are determined with the concept of *domination*. One node n_1 dominates another node n_2 when all paths from start event to n_2 include n_1. A node n_3 postdominates another node n_4 when all paths from n_4 to end event include n_3.

Definition 4 (Process Fragment). *A process fragment of process model G is a subprocess of G defined through a split and a merge gateway g_s and g_m where g_s dominates g_m, g_m postdominates g_s and every loop either contains both g_s and g_m or none of the two. Node n belongs to process fragment defined by (g_s, g_m) when g_s dominates n and g_m postdominates n. Process fragments can be nested when all nodes belonging to fragment f_1 are completely included* *in another fragment f_2. Then, f_1 is a child of f_2. Fragments can be AND- or XOR-fragments, depending on the gateway types defining them.*

One trivial process fragment that is always present is the fragment (start event, end event), the only fragment not defined by gateways. All nodes belonging to this fragment and to no other fragment are on level 0. All other nodes have a level assignment depending on the fragment hierarchy and the smallest fragment they belong to. The level assignment for the activities of the example process of Fig. 1 is the following: level 0: {A} (trivial); level 1: {B, C} (XOR-fragment), {D, E, F} (XOR-fragment); level 2: {G, H} (AND-fragment).

Two activities are *optionally in parallel* if they belong to the same AND-fragment but to different branches of the fragment. Two activities are *mandatorily in parallel* if they belong to the same AND-fragment but to different branches of the fragment and if only other activities or AND-gateways belong to this AND-fragment. Mandatorily in parallel is a stronger property and optionally in parallel. Within a non-loop XOR-fragment, there are at least two exclusive paths from g_s to g_m and in an AND-fragment there are at least two parallel paths from split to merge gateway. When determining the dependencies for two activities, we separately assign existence and ordering dependencies. For the existence dependencies, we need the fragment hierarchy. The ordering dependencies consider pre- and postconditional ordering.

Definition 5 (Control-flow-based Ordering Dependencies in Imperative Process Models). *Let $G = (N, E, \tau, \lambda)$ be an imperative process model and A and B two activities.*

- *We assign a mandatory postconditional ordering dependency to A and B, $\twoheadrightarrow$(A,B), iff B postdominates A.*
- *We assign a mandatory preconditional ordering dependency to A and B, $\twoheadleftarrow$(A,B), iff A dominates B.*
- *We assign an optional postconditional ordering dependency to A and B, $\rightarrow$(A,B), iff there is a chain of edges from A to B but B does not postdominate A.*
- *We assign an optional preconditional ordering dependency to A and B, $\leftarrow$(A,B), iff there is a chain of edges from A to B but A does not dominate B.*
- *If A and B are in parallel (optionally and mandatorily), we assign $\leftrightarrow$(A,B) and $\leftrightarrow$(B,A).*
- *If there is no mandatory or optional ordering possible for A and B, we assign $-$(A,B).*

Note that it is possible to have, for example, $\leftrightarrow$(A,B) but $-$(B,A).

Definition 6 (Control-flow-based Existence Dependencies in Imperative Process Models). *Let $G = (N, E, \tau, \lambda)$ be an imperative process model and A and B two activities.*

- *We assign a symmetric existence dependency to A and B, ==(A,B), iff A dominates B and B postdomiates A or B dominates A and A postdominates B or A and B are mandatorily in parallel.*

- *We assign a non-symmetric existence dependency to A and B, =>(A,B), iff B dominates A but not A postdominates B or B postdominates A but not A dominates B.*
- *We assign an excluding dependency between A and B, ><(A,B), iff A and B belong to the same XOR-fragment but to different branches and there is neither a chain of edges from A to B nor from B to A.*

The excluding dependency ><(A,B) is regarded global; when there is a loop in the model, activities belonging to a child XOR-fragment of the loop XOR-fragment are not exclusive.

Definition 7 (Data-based Dependencies in Imperative Process Models). *Let $G = (N, E, \tau, \lambda)$ be an imperative process model with already identified control-flow-based dependencies, A and B two activities and δ a data identifier. There is a data-based dependency between A and B iff*

- *we do not have ><(A,B) and*
- *we have either $\lambda_2(A) = \delta = \lambda_3(B)$ or $\lambda_3(A) = \delta = \lambda_2(B)$.*

The already identified control-flow-based dependencies are then changed into data-based ones by changing the symbols.

For the example BPMN process model of Fig. 1, the dependencies are shown in a matrix structure in Table 4. Properties of the single activities are not considered here, so the diagonal is empty. The model of Fig. 1 contains information concerning control-flow and data-flow. Note that if a data-based dependency is available for two activities, then an additional control-flow dependency does not have to be entered into the matrix. We regard data-based dependencies as more valuable than control-flow-based dependencies. Existence dependencies are always mirrored at the diagonal.

Comparing these dependencies with related work about behavioral profiles for imperative process models, e.g., [18], we see that we can represent the behavioral profiles with our flow dependencies. But due to the distinction between ordering and existence dependencies, we can specify more different profiles with flow dependencies than with behavioral profiles. For the comparison, it holds: the *strict order relation* is identified with the mandatory ordering, the *exclusiveness relation* with the exclusive dependency, and the *interleaving order relation* with the optional ordering in both directions (cf. [18]). [26] mentions a further relation, namely *co-occurrence relation*, which is represented by the non-symmetric existence dependency in our notation. As a disadvantage of the co-occurrence relation, the authors mention that the co-occurrence relation can only be specified for activities in strict order relation (thus implying strict order relation), it cannot be detected for interleaving order relation. This is not the case for the non-symmetric existence dependency presented in the work at hand.

	A	B	C	D	E	F	G	H
A		${}^{c}{<=}$ ${}^{c}{\twoheadleftarrow\!\!\!\longrightarrow}$	${}^{c}{<=}$ ${}^{c}{\twoheadleftarrow\!\!\!\longrightarrow}$	${}^{c}{<=}$ ${}^{c}{\twoheadleftarrow\!\!\!\longrightarrow}$	${}^{c}{<=}$ ${}^{c}{\twoheadleftarrow\!\!\!\longrightarrow}$	${}^{c}{<=}$ ${}^{c}{\twoheadleftarrow\!\!\!\longrightarrow}$	${}^{c}{<=}$ ${}^{c}{\twoheadleftarrow\!\!\!\longrightarrow}$	${}^{c}{<=}$ ${}^{c}{\twoheadleftarrow\!\!\!\longrightarrow}$
B	${}^{c}{=>}$ $-$		${}^{c}{><}$	${}^{c}{\leftrightarrow}$	${}^{c}{\leftrightarrow}$	${}^{c}{\leftrightarrow}$	${}^{c}{\leftrightarrow}$	${}^{c}{\leftrightarrow}$
C	${}^{c}{=>}$ $-$	${}^{c}{><}$		${}^{c}{\leftrightarrow}$	${}^{c}{\leftrightarrow}$	${}^{c}{\leftrightarrow}$	${}^{c}{\leftrightarrow}$	${}^{c}{\leftrightarrow}$
D	${}^{c}{=>}$ $-$	$-$	$-$		${}^{c}{><}$	${}^{c}{><}$	${}^{c}{><}$	${}^{c}{><}$
E	${}^{c}{=>}$ $-$	$-$	$-$	${}^{c}{><}$		${}^{d}{==}$ ${}^{d}{\twoheadleftarrow\!\!\!\twoheadrightarrow}$	${}^{c}{==}$ ${}^{c}{\twoheadleftarrow\!\!\!\twoheadrightarrow}$	${}^{c}{==}$ ${}^{c}{\twoheadleftarrow\!\!\!\twoheadrightarrow}$
F	${}^{c}{=>}$ $-$	$-$	$-$	${}^{c}{><}$	${}^{d}{==}$ $-$		${}^{c}{==}$ ${}^{c}{\twoheadleftarrow\!\!\!\twoheadrightarrow}$	${}^{c}{==}$ ${}^{c}{\twoheadleftarrow\!\!\!\twoheadrightarrow}$
G	${}^{c}{=>}$ $-$	$-$	$-$	${}^{c}{><}$	${}^{c}{==}$ $-$	${}^{c}{==}$ $-$		${}^{c}{==}$ ${}^{c}{\leftrightarrow}$
H	${}^{c}{=>}$ $-$	$-$	$-$	${}^{c}{><}$	${}^{c}{==}$ $-$	${}^{c}{==}$ $-$	${}^{c}{==}$ ${}^{c}{\leftrightarrow}$	

Table 4. Existence and ordering dependencies according to Definitions 5, 6, and 7 describing the example model of Fig. 1.

3.2 Dependencies in Declarative Process Models

Like for imperative process models, we now show how to derive flow dependencies from declarative process models. In conformance with the general approach of declarative process modeling, the default ordering of two activities is an optional ordering in both directions and the default existence is not specified. The dependencies are assigned according to the list of rules given in Table 2. For the control-flow rules, the dependencies are directly assigned as stated in Definition 8.

Definition 8 (Control-flow-based Dependencies in Declarative Process Models). *Let $D = (\mathcal{A}, \mathcal{R})$ be a declarative process model. For control-flow rules, the following dependencies hold:*

- *s/c of A at $t \Rightarrow$ s/c of B at s: $=>$(A,B) and $\leftrightarrow$(A,B) and $\leftrightarrow$(B,A)*
- *s/c of A at $t \Rightarrow$ s/c of B at $s < t$: $=>$(A,B) and $\leftrightarrow$(A,B) and $\twoheadleftarrow\!\!\!\longrightarrow$(B,A)*
- *s/c of A at $t \Rightarrow$ s/c of B at $s > t$: $=>$(A,B) and $\longleftarrow\!\!\!\twoheadrightarrow$(A,B) and $\leftrightarrow$(B,A)*
- *s/c of A at $t \Rightarrow$ not s/c of B at s: $><$(A,B) and $-$(A,B) and $-$(B,A)*
- *s/c of A at $t \Rightarrow$ not s/c of B at $s < t$: $\leftrightarrow$(A,B) and $-$(B,A)*
- *s/c of A at $t \Rightarrow$ not s/c of B at $s > t$: $-$(A,B) and $\leftrightarrow$(B,A)*

A symmetric existence connection between two activities is introduced the following way: $=>$(A,B) $\wedge$ $=>$(B,A) $\Rightarrow$ $==$(A,B). When two rules affect the same activity tuple, $\twoheadrightarrow$ and $-$ predominate the default $\rightarrow$.

Due to transitivities in the flow dependencies, e.g., $=>$(A,B) and $=>$(B,C) lead to $=>$(A,C), it is possible to derive dependencies not only directly from process rules as stated in Definition 8 but to get more dependencies in a second, iterative step. Note that transitivity within flow dependencies is different from transitivity in data rules stated in Definition 9. The assignment of data-based dependencies needs one derivation step performed on the process rules to get dependencies between two activities instead of two data objects or an activity and a data object.

Definition 9 (Data-based Dependencies in Declarative Process Models). *Let $D = (\mathcal{A}, \mathcal{R})$ be a declarative process model and $\mathcal{R}_d \subseteq \mathcal{R}$ all data rules. The data rules are traced back to control-flow rules of the form presumed in Definition 8 applying transitivity of the $<$ operator for the time information and natural deduction calculus. Then, the data-flow dependencies between two activities can be derived in the same way as the control-flow dependencies.*

As an illustration of Definition 9, regard rules 4 and 8 of the example model of Table 1. Assume activity F is executed at time t (F_t). The natural deduction calculus yields the following result: F_t together with rule 4 results in write of δ at $s < t$ ($\delta_s \wedge s < t$) applying modus ponens. Elimination out of conjunction yields δ_s. Together with rule 8 (write of δ at $s \Rightarrow$ not complete of D at r, which implies the statement write of $\delta_s \Rightarrow$ not complete of D_r) and modus ponens, we get $(F_t \wedge s < t) \Rightarrow \neg D_r$ which is the same as $F \Rightarrow \neg D$ as D is independent of any execution time and t is arbitrary. Thus, it holds s/c of F at $t \Rightarrow$ not s/c of D at s. The derived dependency is therefore ${}^{d}{><}\ (F, D)$.

For the example of Table 1, the dependency matrix is given in Table 5. Note that one of the rules 2 and 3 is redundant. The shown dependencies are the directly derivable ones and that ones achieved through transitivities within the directly derivable flow dependencies (not the transitivities within the data rules). In the example model, only the dependency between D and H is achieved through flow dependency transitivity. As the derivation is not fully data-based, it is a control-flow dependency. A list of all possibilities for flow dependency transitivity is given in Table 6. The table reads as follows: when we derived a non-default flow dependency listed in column "dependency 1", actually an existence and a certain combination of ordering dependencies, and a dependency listed in column "dependency 2" on the same row, we have to refine, i.e., restrict, a third dependency according to the third column of the same row. For the ordering dependencies, there are often several combination possibilities that have to be distinguished. Also remember that ==(A,B) includes both =>(A,B) and =>(B,A). In the example model, the transitive flow dependency between D and H is achieved via the transitivity ${}^{c}{=>}$(H,G) and ${}^{c}{\leftrightarrow}$(H,G) and ${}^{c}{\leftrightarrow}$ (G,H) together with ${}^{d}{><}$(G,D) which leads to ${}^{c}{><}$(H,D) according to the second possibility mentioned in Table 6. In the not specified cells, i.e., all those that cannot be refined, we entered the default values in Table 5.

4 Comparison of Two Process Models

When comparing two models, in this case their behavior, we can now compare the two dependency matrices. For this, we determine a hierarchy of the de-

	A	B	C	D	E	F	G	H
A		$^{c}{<=}$ $^{c}\twoheadleftarrow\!\!\longrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$
B	$^{c}{=>}$ $^{c}\leftrightarrow$		$^{c}{><}$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$
C	$^{c}\leftrightarrow$	$^{c}{><}$		$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$
D	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$		$^{d}{><}$	$^{d}{><}$	$^{d}{><}$	$^{c}{><}$
E	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{d}{><}$		$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$
F	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{d}{><}$	$^{c}\leftrightarrow$		$^{c}\leftrightarrow$	$^{c}\leftrightarrow$
G	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{d}{><}$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$		$^{c}{<=}$ $^{c}\leftrightarrow$
H	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}{><}$	$^{c}\leftrightarrow$	$^{c}\leftrightarrow$	$^{c}{=>}$ $^{c}\leftrightarrow$	

Table 5. Flow dependencies according to Definitions 8 and 9 in the example model of Table 1.

pendencies. This allows us to state whether two models show contradictionary behavior, similar behavior, or are related in a way that one model is overspecified/underspecified compared to the other model.

We set the hierarchy of the dependencies according to the degree of restrictions the respective dependencies impose. With $dep1 \prec dep2$ we denote that $dep1$ is more restrictive regarding execution possibilities than $dep2$, i.e., the set of possible executions induced by $dep1$ is a proper subset of those imposed by $dep2$. Thus, $\prec$ is transitive. With $dep1 \,\#\, dep2$ we denote that $dep1$ and $dep2$ impose contradictional behavior (the intersection of both sets is empty). Two dependencies are not comparable when their intersection and their relative complements are not empty. With *nspec* we denote the absence of an existence dependency. We get the following hierarchy for the existence dependencies:

$$== \prec => \prec \; nspec$$

Comparing $=>$ and $<=$ is not possible, i.e., we assign a $\circ$ for marking the non-comparability. Here, we can possibly derive a more meaningful statement when looking at the comparison of the ordering dependencies. For the non-existence dependency we get

$$>< \;\#\; == .$$

The dependencies $><$ and $=>$ resp. *nspec* are not comparable in the first place regarding execution possibilities. However, when considering contradictions on basis of musts and prohibitions more severe than commonalities in allowed (but not prescribed) possibilities, we can add

$$=> \;\#\; >< .$$

dependency 1	dependency 2	derived dependency
=>(A,B)	=>(B,C)	=>(A,C)
and ↞⟶(B,A) and ↔(A,B)	and ↞⟶(C,B) and ↔(B,C)	and ↞⟶(C,A) and ↔(A,C)
and ⟵↠(A,B) and ↔(B,A)	and ⟵↠(B,C) and ↔(C,B)	and ⟵↠(A,C) and ↔(A,C)
and any other combinations of ordering dependencies		and ↔(A,C) and ↔(C,A)
=>(A,B)	><(B,C)	><(A,C)
and arbitrary ordering		
=>(A,B)		
and ⟵↠(A,B) and ↔(B,A)	↔(B,C) and −(C,B)	↔(A,C) and −(C,A)
=>(A,B)		
and ↔(A,B) and ↞⟶(B,A)	↔(C,B) and −(B,C)	↔(C,A) and −(A,C)
↔(A,B) and −(B,A)	↔(B,C) and −(C,B)	↔(A,C) and −(C,A)

Table 6. Transitive flow dependency derivation.

>< and *nspec* are not comparable since a non-specified existence dependency does not demand for a certain execution prohibited by ><.

For the ordering dependencies, we have three parallel hierarchies:

$$\begin{tikzcd}
 & \leftrightarrow \arrow[dl, dash, "\curlyvee"'] \arrow[d, dash, "\curlyvee"'] \arrow[dr, dash, "\curlyvee"] & \\
- & \text{⟵↠} \arrow[d, dash, "\curlyvee"'] & \text{↞⟶} \arrow[dl, dash, "\curlyvee"'] \\
 & \text{↞↠} &
\end{tikzcd}$$

Furthermore, it is

$$- \;\#\; \text{↞↠}, \quad - \;\#\; \text{⟵↠}, \quad \text{and} \quad - \;\#\; \text{↞⟶}$$

because the mandatory ordering requires an execution order that is not possible for the not existing ordering dependency −. Ordering dependencies ⟵↠ and ↞⟶ are not comparable.

This hierarchy allows us to compare the two example models, i.e., to set up a comparison matrix on basis of the dependency matrices of Tables 4 and 5. The comparison matrix is shown in Table 7 where the imperative dependencies are mentioned before the declarative ones. The equal sign = means that the dependencies are the same at the respective positions in the dependency matrices. We do not distinguish between control-flow and data dependencies here as this does not affect the kind of relation of the two models. It affects, however, the degree of the respective relation, i.e., the degree of similarity. Existence and

	A	B	C	D	E	F	G	H
A		$=$ $=$	$\prec$ $\prec$	$\prec$ $\prec$	$\prec$ $\prec$	$\prec$ $\prec$	$\prec$ $\prec$	$\prec$ $\prec$
B	$\prec$ $\prec$		$=$	$=$ $=$	$=$ $=$	$=$ $=$	$=$ $=$	$=$ $=$
C	$\prec$ $\prec$	$=$		$=$ $\prec$	$=$ $\prec$	$=$ $\prec$	$=$ $\prec$	$=$ $\prec$
D	$\prec$ $\prec$	$=$ $\prec$	$=$ $\prec$		$=$	$=$	$=$	$=$
E	$\prec$ $\prec$	$=$ $\prec$	$=$ $\prec$	$=$		$\prec$ $\prec$	$\prec$ $\prec$	$\prec$ $\prec$
F	$\prec$ $\prec$	$=$ $\prec$	$=$ $\prec$	$=$	$\prec$ $\prec$		$\prec$ $\prec$	$\prec$ $\prec$
G	$\prec$ $\prec$	$=$ $\prec$	$=$ $\prec$	$=$	$\prec$ $\prec$	$\prec$ $\prec$		$\prec$ $=$
H	$\prec$ $\prec$	$=$ $\prec$	$=$ $\prec$	$=$	$\prec$ $\prec$	$\prec$ $\prec$	$\prec$ $=$	

Table 7. Comparison of the two example process models.

ordering dependencies are compared separately, which is why we get up to two comparison signs in each matrix cell.

We see that the two example models are in a subsumption relation. The declarative model is more general than the imperative model, i.e., it allows for more execution possibilities than the imperative model and the imperative model is completely reproducable by the declarative one. There are no contradictions in the comparison of the two models. When comparing a declarative and an imperative process model, we can expect that if there is a subsumption relation between the two models, it is of the form that the imperative model is part of the declarative one because the declarative model usually allows for a more flexible execution. If there are any contradictions, we can locate them precisely, i.e., the activities and the relations between them through having a closer look at the dependency matrices.

5 Limitations and Future Work

Like similar approaches in related work, the method described in the paper at hand determines flow dependencies for every two activities of a process model. However, declarative modeling languages sometimes allow for constraints involving more than two activities. These are not captured in the presented approach. Thus, it is an issue of future work to expand the definition of declarative process models, i.e., the set of constraint templates. This extension also involves the inclusion of process rules including agents and non-human resources.

Another application that makes use of the presented approach is to derive a similarity measure, i.e., a number between 0 and 1 to specify the degree of similarity of two compared models. This similarity measure could use the dependency hierarchies given in Section 4, e.g., the distance between two dependencies when they are in a hierarchy. Contradictions would of course lead to a similarity of 0. Also, the information whether a dependency is control-flow based or data-based is useful for determining similarity. With a similarity measure, it is not only possible to state whether two models are in a subsumption relation but also how similar they are, i.e., how great their shared behavior is related to all possible behavior of the subsuming model. Or, if they are contradictionary, the extent of their difference. For large models, it is easier to judge their similarity at a first glance according to one approximating number than looking at a complex table.

The presented dependency approach regards the behavior of a process model, which is just one aspect of a process. It should be combined with a comparison or similarity determination of the other perspectives as well, like agents, data, non-human resources, and activity descriptions. Note that this data similarity does not use the same information used for the data-based flow dependencies shown in this paper. It also requires the adaption of existing matching methods for imperative models to also work with declarative models.

References

1. van der Aalst, W., Alves de Medeiros, A., Weijters, A.: Process equivalence: Comparing two process models based on observed behavior. In: Dustdar, S., Fiadeiro, J., Sheth, A. (eds.) Business Process Management, LNCS, vol. 4102, pp. 129–144. Springer Berlin Heidelberg (2006)
2. Bae, J., Caverlee, J., Liu, L., Yan, H.: Process mining by measuring process block similarity. In: Eder, J., Dustdar, S. (eds.) Business Process Management Workshops: Proceedings. pp. 141–152. Springer Berlin Heidelberg (2006)
3. Baumann, M.H., Baumann, M., Schönig, S., Jablonski, S.: Towards multi-perspective process model similarity matching. In: Barjis, J., Pergl, R. (eds.) Enterprise and Organizational Modeling and Simulation, LNBIP, vol. 191, pp. 21–37. Springer Berlin Heidelberg (2014)
4. Baumann, M., Baumann, M.H., Ackermann, L., Schönig, S., Jablonski, S.: Ansätze zum ähnlichkeitsabgleich von deklarativen geschätsprozessmodellen `https://epub.uni-bayreuth.de/2534/`, preprint
5. Baumann, M., Baumann, M.H., Jablonski, S.: On behavioral process model similarity matching: A centroid-based approach 5, 125–131 (2015)
6. Baumann, M., Baumann, M.H., Schönig, S., Jablonski, S.: Resource-aware process model similarity matching. In: Toumani, F., et al. (eds.) Service-Oriented Computing - ICSOC 2014 Workshops, Lecture Notes in Computer Science, vol. 8954, pp. 96–107. Springer International Publishing (2015)
7. Becker, J., Delfmann, P., Dietrich, H.A., Steinhorst, M., Eggert, M.: Business process compliance checking – applying and evaluating a generic pattern matching approach for conceptual models in the financial sector. Information Systems Frontiers 18(2), 359–405 (2014)
8. Becker, M., Laue, R.: A comparative survey of business process similarity measures. Computers in Industry 63(2), 148–67 (2012)
9. Dijkman, R., Dumas, M., van Dongen, B., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Information Systems 36(2), 498 – 516 (2011)

10. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) Business Process Management, LNCS, vol. 5701, pp. 48–63. Springer Berlin Heidelberg (2009)
11. Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., Zbyslaw, A.: Freeflow: Mediating between representation and action in workflow systems. In: Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work. pp. 190–198. CSCW '96, ACM (1996)
12. Dumas, M., García-Bañuelos, L., Dijkman, R.M.: Similarity search of business process models. IEEE Data Eng. Bull. 32(3), 23–28 (2009)
13. Ekanayake, C.C., Dumas, M., García-Bañuelos, L., La Rosa, M., ter Hofstede, A.H.M.: Approximate clone detection in repositories of business process models. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012: Proceedings, pp. 302–318. Springer Berlin Heidelberg (2012)
14. Fellmann, M., Delfmann, P., Koschmider, A., Laue, R., Leopold, H., Schoknecht, A.: Semantic technology in business process modeling and analysis. part 1: Matching, modeling support, correctness and compliance. EMISA Forum 35(1) (2015)
15. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: The provop approach. Journal of Software Maintenance and Evolution: Research and Practice 22(6-7), 519–546 (2010)
16. Jablonski, S., Bussler, C.: Workflow management: modeling concepts, architecture and implementation. International Thomson Computer Press (1996)
17. Klinkmüller, C., Leopold, H., Weber, I., Mendling, J., Ludwig, A.: Listen to me: Improving process model matching through user feedback. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) Business Process Management: Proceedings. pp. 84–100. Springer International Publishing (2014)
18. Kunze, M., Weidlich, M., Weske, M.: Behavioral similarity – a proper metric. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. Proceedings. pp. 166–181. Springer Berlin Heidelberg (2011)
19. Montali, M.: Specification and Verification of Declarative Open Interaction Models – A Logic-based framework. Ph.D. thesis, Alma Mater Studiorum Universitá di Bologna (2009)
20. Pesic, M., Schonenberg, H., van der Aalst, W.: DECLARE: Full support for loosely-structured processes. In: Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International. pp. 287–287 (2007)
21. Pesic, M., van der Aalst, W.M.: Towards a reference model for work distribution in workflow management systems. Business Process Reference Models pp. 30–44 (2005)
22. Pittke, F., Leopold, H., Mendling, J., Tamm, G.: Enabling reuse of process models through the detection of similar process parts. In: La Rosa, M., Soffer, P. (eds.) Business Process Management Workshops, LNBIP, vol. 132, pp. 586–597. Springer Berlin Heidelberg (2013)
23. Polyvyanyy, A., Smirnov, S., Weske, M.: On application of structural decomposition for process model abstraction. In: BPSC. pp. 110–122. Citeseer (2009)
24. Tealeb, A., Awad, A., Galal-Edeen, G.: Context-based variant generation of business process models. In: Bider, I., et al. (eds.) Enterprise, Business-Process and Information Systems Modeling: Proceedings, pp. 363–377. Springer Berlin Heidelberg (2014)
25. Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J.: Process compliance measurement based on behavioural profiles. In: Pernici, B. (ed.) Advanced Information

Systems Engineering: Caise 2010 Proceedings. pp. 499–514. Springer Berlin Heidelberg (2010)
26. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient computation of causal behavioural profiles using structural decomposition. In: Lilius, J., Penczek, W. (eds.) Applications and Theory of Petri Nets: 31st International Conference, PETRI NETS 2010, Braga, Portugal, June 21-25, 2010. Proceedings. pp. 63–83. Springer Berlin Heidelberg (2010)
27. Wombacher, A.: Evaluation of technical measures for workflow similarity based on a pilot study. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, LNCS, vol. 4275, pp. 255–272. Springer Berlin Heidelberg (2006)
28. Xing, J., Zhang, X., Song, W., Yang, Q., Ge, J., Wang, H.: Bpel similarity – a metric based on activity constraint graphs. In: Song, M., Wynn, M.T., Liu, J. (eds.) AP-BPM 2013. Selected Papers. pp. 39–55. Springer International Publishing (2013)
29. Zugal, S., Pinggera, J., Weber, B.: Creating declarative process models using test driven modeling suite. In: Nurcan, S. (ed.) IS Olympics: Information Systems in a Diverse World, LNBIP, vol. 107, pp. 16–32. Springer Berlin Heidelberg (2012)