

# Comparing Imperative and Declarative Process Models

Michaela Baumann

Institute for 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. 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 many use cases mentioned in literature, cf. [1], [7], [12]. On the one hand, it is often referred to in the context of repository management. When a repository grows, the identification of duplicate models [11] and model variants [22], or the reuse of model parts [20] becomes a challenging issue. On the other hand, also the comparison of models with reference models [19] for verifying compliance is one application field of measuring similarity of process models. The method is thereby usually split into two steps: first, a mapping is established between the two models that shall be compared. Second, a similarity value is computed based on the mapping. This procedure is iterated, to find the best mapping that provides the highest similarity value. The similarity value can include several aspects, referred to as the five perspectives [14] of a process model: task description, control-flow, data-flow, human/non-human resources and operational issues. For process models represented 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], [8], [9]. 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], [8], for dependency graphs [2], or for abstracted, graph-like process models, e.g., [3], [15]. 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 [24] 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 suggested 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. 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 [18] but also impose a difficulty to fully understand the models [26]. 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* [13]. This approach is similar to the behavioral profile techniques proposed, for example, in [2], [16], [25], 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 the remainder of this section, imperative and declarative process models are defined. Section 2 describes how we can extract flow dependencies from both imperative and declarative process models. With help of a hierarchy of the dependencies, Section 3 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 4 concludes the paper.

In order to abstract from a specific imperative process modeling language, we regard imperative process models as business process graphs like, e.g., in [9].

**Definition 1 (Imperative Process Model).** *A (graphical) imperative process model  $G$  is a tuple  $(N, E, \tau, \lambda)$  where*

- $N$  is a set of nodes,  $E \subseteq N^2$  is a set of edges,  $\tau : N \rightarrow \mathcal{T}$  is a function mapping nodes to types,  $\lambda : N \rightarrow \mathcal{L}$  is a function mapping nodes to labels.

- $(N, E)$  is a connected graph.
- We define five different types of nodes:  $\mathcal{T} = \{\text{start event, end event, activity, AND-gateway, XOR-gateway}\}$ .
  - There is exactly one start event and one end event.
  - Function  $\lambda$  is a three-dimensional mapping that maps each node to a task description ( $\lambda_1$ ), a set of data identifiers for incoming data ( $\lambda_2$ ), and a set of data identifiers for outgoing data ( $\lambda_3$ ).
  - Splits and merges are only possible with gateways.
  - The model has to be sound [21].

Data objects are represented by their data identifiers stored in set  $\mathcal{D}$ . Definition 1 allows for process models with loops, modeled with XOR-gateways. Fig. 1 shows an imperative process model in BPMN with 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  $\delta$  being the data identifier in  $\mathcal{D}$ . Declarative process models are of the form defined in Definition 2. This definition is in accordance with [17].

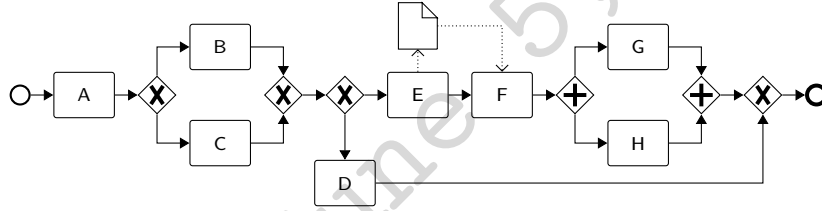


Fig. 1. Example of an imperative process model (BPMN model).

**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. Each activity has at least two possible 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,  $\mathcal{R}_d \subseteq \mathcal{R}$ ).

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. A list of all possible rules enabled through Definition 2 is provided in Table 1 regarding two events: start and complete; and an exemplary declarative process model is given

in Table 2. Variables  $t$  and  $s$  are points of time and  $\delta$  denotes a specific data object. Rules 1, 2, 3, and 7 are each connecting two activities 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 independent, i.e., they hold for  $s \leq t$  and for  $s > t$ .

control-flow rules	data rules
s/c of A at $t \Rightarrow$ (not) s/c of B at $s(\leq t)$	s/c of A at $t \Rightarrow$ (not) write of $\delta$ at $s(\leq t)$ write of $\delta$ at $t \Rightarrow$ (not) s/c of B at $s(\leq t)$ write of $\delta$ at $t \Rightarrow$ (not) write of $\varepsilon$ at $s(\leq t)$

**Table 1.** List of all 24 possible rules in a declarative process model with A and B denoting activities and  $\delta$  and  $\varepsilon$  data identifiers; s/c stands for either start or complete;  $t$  and  $s$  are time variables.

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 $\delta$ at $s < t$	“F can only be started when data $\delta$ is already available, i.e., F consumes $\delta$ ”
5	complete of E at $t \Rightarrow$ write of $\delta$ at $s$	“The completion of E requires that data $\delta$ is written, i.e., E causes the production of $\delta$ ”
6	start of G at $t \Rightarrow$ write of $\delta$ 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 $\delta$ at $t \Rightarrow$ not complete of D at $s$	“As soon as $\delta$ is produced, D may not be completed”

**Table 2.** Example of a declarative process model with rule explanations.

## 2 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 [25] for an example). In the work at hand, we exemplarily show this for the data perspective.

**Definition 3 (Dependency Type).** *We distinguish 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 Table 3.

	control-flow	data
mandatory ordering	$c \rightarrow$	$d \rightarrow$
optional ordering	$c \dashrightarrow$	$d \dashrightarrow$
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  $\rightarrow(A,B)$  means that after A, B has to be executed. A mandatory preconditional ordering  $\leftarrow(A,B)$  means that before B, A has to be executed. It is not possible to have both  $\rightarrow(A,B)$  and  $\rightarrow(B,A)$  as this would be an infinite cycle. An optional postconditional ordering  $\dashrightarrow(A,B)$  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  $\dashleftarrow(A,B)$  means that it is possible but not necessary to do A before B. Ordering arrows for one tuple of activities are combined, e.g.,  $\longleftrightarrow(A,B)$ . 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  $\leftarrow(A,B)$ . Thus, for each activity tuple we have five different kinds of ordering dependencies:  $\leftrightarrow(A,B)$ ,  $\longleftrightarrow(A,B)$ ,  $\dashrightarrow(A,B)$ ,  $\dashleftarrow(A,B)$ , and  $-(A,B)$ . 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 ( $\rightarrow(A,B)$  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 [10].

In the following, we show how to derive *flow dependencies in imperative process models*. To assign the correct dependencies to every pair of activities, we analyze the structure of the process model as done in [21]. 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  $g_m$  of the same type 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, e.g.,  $f_1$  is a child of  $f_2$ . Fragments can be AND- or XOR-fragments, depending on the gateway types defining them.

The trivial process fragment is (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, level 1: B, C, E, E, F, level 2: G, H. 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 non of them belongs to a branch between a XOR-split and the corresponding XOR-merge within the AND-fragment. 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. Note that it is possible to have, for example,  $\leftrightarrow(A,B)$  but  $-(B,A)$ .

**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  $B$ ,  $\rightarrow(A,B)$ , iff  $B$  postdominates  $A$ .
- We assign a mandatory preconditional ordering dependency to  $A$  and  $B$ ,  $\leftarrow(A,B)$ , 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  $\leftrightarrow(A,B)$ .
- If there is no mandatory or optional ordering possible for  $A$  and  $B$ , we assign  $-(A,B)$ .

**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  $B$  postdominates  $A$ ,  $B$  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. When there is no existence dependency given, we assign  $\sim$ .

**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  $\delta$  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 in Table 4. The model of Fig. 1 contains information concerning control-flow and data-flow. 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., [16], [23], 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. In [23], the authors mention that the so-called co-occurrence relation can only be specified for activities in strict order relation and that 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.

Like for imperative process models, we now show how to derive *flow dependencies in 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 ( $\leftrightarrow$ ) and the default existence is not specified ( $\sim$ ). For the control-flow rules of Table 1, the dependencies are assigned as stated in Definition 8.

**Definition 8 (Control Flow based Dependencies in Declarative Process Models).** Let  $D = (A, \mathcal{R})$  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  $\leftarrow(B,A)$
- $s/c$  of  $A$  at  $t \Rightarrow s/c$  of  $B$  at  $s > t$ :  $=>(A,B)$  and  $\leftarrow(A,B)$  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:  $\Rightarrow(A,B) \wedge \Rightarrow(B,A) \Rightarrow \Rightarrow(A,B)$ . When two rules affect the same activity tuple,  $\Rightarrow$  and  $-$  predominate the default  $\rightarrow$ .

Due to transitivities, e.g.,  $\Rightarrow(A,B)$  and  $\Rightarrow(B,C)$  lead to  $\Rightarrow(A,C)$ , it is possible to derive dependencies not only directly from process rules given in Definition 8 but to get more dependencies in iterative steps.

	A	B	C	D	E	F	G	H
A		$c<=<$ $c\leftrightarrow$	$c<=<$ $c\leftrightarrow$	$c<=<$ $c\leftrightarrow$	$c<=<$ $c\leftrightarrow$	$c<=<$ $c\leftrightarrow$	$c<=<$ $c\leftrightarrow$	$c<=<$ $c\leftrightarrow$
B	$c=>$ $-$		$c><$	$\sim$ $c\leftrightarrow$	$\sim$ $c\leftrightarrow$	$\sim$ $c\leftrightarrow$	$\sim$ $c\leftrightarrow$	$\sim$ $c\leftrightarrow$
C	$c=>$ $-$	$c><$		$\sim$ $c\leftrightarrow$	$\sim$ $c\leftrightarrow$	$\sim$ $c\leftrightarrow$	$\sim$ $c\leftrightarrow$	$\sim$ $c\leftrightarrow$
D	$c=>$ $-$	$\sim$ $-$	$\sim$ $-$		$c><$	$c><$	$c><$	$c><$
E	$c=>$ $-$	$\sim$ $-$	$\sim$ $-$	$c><$		$d==$ $d\leftrightarrow$	$c==$ $c\leftrightarrow$	$c==$ $c\leftrightarrow$
F	$c=>$ $-$	$\sim$ $-$	$\sim$ $-$	$c><$	$d==$ $-$		$c==$ $c\leftrightarrow$	$c==$ $c\leftrightarrow$
G	$c=>$ $-$	$\sim$ $-$	$\sim$ $-$	$c><$	$e==$ $-$	$e==$ $-$		$c==$ $c\leftrightarrow$
H	$c=>$ $-$	$\sim$ $-$	$\sim$ $-$	$c><$	$e==$ $-$	$c==$ $-$	$c==$ $c\leftrightarrow$	

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

**Definition 9 (Data-based Dependencies in Declarative Process Models).** Let  $D = (A, \mathcal{R})$  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 2. 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  $\delta$  at  $s < t$  ( $\delta_s \wedge s < t$ ) applying modus ponens. Elimination out of conjunction yields  $\delta_s$ . Together with rule 8 (write of  $\delta$  at  $s \Rightarrow$  not complete of D at  $r$ , which implies 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 2, the dependency matrix is given in Table 5. A list of all possibilities for flow dependency transitivity is given in Table 6. The absence of any existence dependency is denoted with  $\sim$ .



	A	B	C	D	E	F	G	H
A		$\overset{c}{\leftarrow} \overset{c}{=}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$
B	$\overset{c}{\rightleftarrows}$		$\overset{c}{>} \overset{c}{<}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$
C	$\overset{c}{\sim}$	$\overset{c}{>} \overset{c}{<}$		$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$
D	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$		$\overset{d}{>} \overset{d}{<}$	$\overset{d}{>} \overset{d}{<}$	$\overset{d}{>} \overset{d}{<}$	$\overset{c}{>} \overset{c}{<}$
E	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{d}{>} \overset{d}{<}$		$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$
F	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{d}{>} \overset{d}{<}$	$\overset{c}{\sim}$		$\overset{c}{\sim}$	$\overset{c}{\sim}$
G	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{d}{>} \overset{d}{<}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$		$\overset{c}{\leftarrow} \overset{c}{=}$
H	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{>} \overset{c}{<}$	$\overset{c}{\sim}$	$\overset{c}{\sim}$	$\overset{c}{\rightleftarrows}$	

**Table 5.** Flow dependencies according to Definitions 8 and 9 in the example model of Table 2.

dependency 1	dependency 2	derived dependency
$\Rightarrow(A,B)$	$\Rightarrow(B,C)$	$\Rightarrow(A,C)$
and $\leftarrow(B,A)$ and $\leftrightarrow(A,B)$	and $\leftarrow(C,B)$ and $\leftrightarrow(B,C)$	and $\leftarrow(C,A)$ and $\leftrightarrow(A,C)$
and $\leftarrow(A,B)$ and $\leftrightarrow(B,A)$	and $\leftarrow(B,C)$ and $\leftrightarrow(C,B)$	and $\leftarrow(A,C)$ and $\leftrightarrow(A,C)$
and any other combinations of ordering dependencies		and $\leftrightarrow(A,C)$ and $\leftrightarrow(C,A)$
$\Rightarrow(A,B)$	$><(B,C)$	$><(A,C)$
and arbitrary ordering		
$\Rightarrow(A,B)$		
and $\leftarrow(A,B)$ and $\leftrightarrow(B,A)$	$\leftrightarrow(B,C)$ and $-(C,B)$	$\leftrightarrow(A,C)$ and $-(C,A)$
and $\leftrightarrow(A,B)$ and $\leftarrow(B,A)$	$\leftrightarrow(C,B)$ and $-(B,C)$	$\leftrightarrow(C,A)$ and $-(A,C)$
$\leftrightarrow(A,B)$ and $-(B,A)$	$\leftrightarrow(B,C)$ and $-(C,B)$	$\leftrightarrow(A,C)$ and $-(C,A)$

**Table 6.** Transitive flow dependency derivation.

### 3 Comparison of Two Process Models

When comparing two models concerning their behavior we can now compare the two dependency matrices. For this, we determine a hierarchy of the dependencies. This allows us to state whether two models show contradictory behavior, similar behavior, or are related in a way that one model is overspecified/underspecified compared to the other model. A 1-to-1 mapping between equal, or at least similar enough activities is assumed to be given. We set the hierarchy of the dependencies according to the degree of restrictions the 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 contradictory behavior (the intersection of both sets is empty). Two dependencies are not comparable ( $\circ$ ).

when their intersection and their relative complements are not empty. We get the following hierarchy for the existence dependencies:  $== \prec => \prec \sim$ . Comparing  $==>$  and  $<=$  is not possible, i.e., we assign  $\circ$  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.  $\sim$  are not comparable in the first place regarding execution possibilities. However, when considering contradictions on basis of musts and prohibitions more severely than commonalities in allowed (but not prescribed) possibilities, we can add  $==> \# ><$ .  $><$  and  $\sim$  are not comparable since a non-specified existence dependency does not demand for a certain execution prohibited by  $><$ . For the ordering dependencies, we have these hierarchies:  $- \prec \leftrightarrow, \leftarrow \prec \leftarrow \prec \leftrightarrow, \leftarrow \prec \leftarrow \prec \leftrightarrow$ . Furthermore, it is  $- \# \leftarrow$ ,  $- \# \leftarrow$ , and  $- \# \leftarrow$  because the mandatory ordering requires an execution order that is not possible for the not existing ordering dependency  $-$ . Ordering dependencies  $\leftarrow$  and  $\leftarrow$  are not comparable. This hierarchy allows us 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. Existence and ordering dependencies are compared separately, which is why we get two comparison signs in each cell. When  $><$  is compared to an existence and ordering relation combination, we assign two  $\#$  symbols. When  $><$  is compared to  $><$ , we assign two  $=$  symbols. 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. There are no contradictions in the comparison of the two models. If there were any contradictions, we could locate them precisely, i.e., the activities and the relations between them through having a closer look at the dependency matrices.

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

**Table 7.** Comparison of the two example process models.

## 4 Limitations and Future Work

It is an issue of future work to expand the definition of declarative process models to allow, e.g., for constraints involving more than two activities and process rules including agents and non-human resources. Also, the extraction of flow dependencies from, e.g., DECLARE models or DCR Graphs is a future issue. 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, since 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 information whether a dependency is control flow based or data-based is useful for determining similarity. The presented dependency approach should be combined with other process perspectives like agents, data, non-human resources, and activity descriptions [3]. Note that this data similarity does not use the same information used for the data-based flow dependencies shown in this paper. An assessment of the presented approach must be conducted as soon as other methods for comparing imperative and declarative models are available.

## References

1. van der Aalst, W., Alves de Medeiros, A., Weijters, A.: Process equivalence: Comparing two process models based on observed behavior. In: *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: *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: *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 Geschäftsprozessmodellen. In: *INFORMATIK 2016: LNI*, vol. 259, pp. 733–738 (2016)
5. Baumann, M., Baumann, M.H., Jablonski, S.: On behavioral process model similarity matching: A centroid-based approach. In: *ICCGI*. vol. 5, pp. 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, M., Laue, R.: A comparative survey of business process similarity measures. *Computers in Industry* 63(2), 148–67 (2012)
8. 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)
9. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In: *Business Process Management, LNCS*, vol. 5701, pp. 48–63. Springer Berlin Heidelberg (2009)

10. 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)
11. 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: BPM 2012: Proceedings, pp. 302–318. Springer Berlin Heidelberg (2012)
12. 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)
13. Jablonski, S.: Do we really know how to support processes? considerations and reconstruction. In: Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., Westfechtel, B. (eds.) Graph Transformations and Model-Driven Engineering, LNCS, vol. 5765, pp. 393–410. Springer Berlin Heidelberg (2010)
14. Jablonski, S., Bussler, C.: Workflow management: modeling concepts, architecture and implementation. International Thomson Computer Press (1996)
15. Klinkmüller, C., Leopold, H., Weber, I., Mendling, J., Ludwig, A.: Listen to me: Improving process model matching through user feedback. In: Business Process Management: Proceedings. pp. 84–100. Springer International Publishing (2014)
16. Kunze, M., Weidlich, M., Weske, M.: Behavioral similarity – a proper metric. In: BPM 2011. Proceedings. pp. 166–181. Springer Berlin Heidelberg (2011)
17. Montali, M.: Specification and Verification of Declarative Open Interaction Models – A Logic-based framework. Ph.D. thesis, Alma Mater Studiorum Università di Bologna (2009)
18. Pestic, 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)
19. Pestic, 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)
20. Pittke, F., Leopold, H., Mendling, J., Tamm, G.: Enabling reuse of process models through the detection of similar process parts. In: Business Process Management Workshops, LNBIP, vol. 132, pp. 586–597. Springer Berlin Heidelberg (2013)
21. Polyvyanyy, A., Smirnov, S., Weske, M.: On application of structural decomposition for process model abstraction. In: BPSC. pp. 110–122. Citeseer (2009)
22. Tealeb, A., Awad, A., Galal-Edeen, G.: Context-based variant generation of business process models. In: Enterprise, Business-Process and Information Systems Modeling: Proceedings, pp. 363–377. Springer Berlin Heidelberg (2014)
23. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient computation of causal behavioural profiles using structural decomposition. In: Applications and Theory of Petri Nets: 31st International Conference, PETRI NETS 2010. pp. 63–83. Springer Berlin Heidelberg (2010)
24. Wombacher, A.: Evaluation of technical measures for workflow similarity based on a pilot study. In: On the Move to Meaningful Internet Systems 2006, LNCS, vol. 4275, pp. 255–272. Springer Berlin Heidelberg (2006)
25. Xing, J., Zhang, X., Song, W., Yang, Q., Ge, J., Wang, H.: Bpel similarity – a metric based on activity constraint graphs. In: AP-BPM 2013. Selected Papers. pp. 39–55. Springer International Publishing (2013)
26. Zugal, S., Pinggera, J., Weber, B.: Creating declarative process models using test driven modeling suite. In: IS Olympics: Information Systems in a Diverse World, LNBIP, vol. 107, pp. 16–32. Springer Berlin Heidelberg (2012)