[MRS+26] J. Michael, B. Rumpe, M. Stachon, S. Stüber, V. Voufo:
Automated Conformance Checking Between Process Models and Their Generalized Reference Process Models.
In: Mining a Scientist's Process: Essays Dedicated to Wil van der Aalst on the Occasion of His 60th Birthday,
Mendling, Jan and Leemans, Sander and van Dongen, Boudewijn F. and Reijers, Hajo (Eds.),
pp. 525–550, LNCS 16480, ISBN 978-3-032-17618-9, DOI 10.1007/978-3-032-17618-9_35, Springer, Mar. 2026.

Automated Conformance Checking Between Process Models and Their Generalized Reference Process Models

Judith Michael[1], Bernhard Rumpe[2], Max Stachon[2(✉)], Sebastian Stüber[2], and Valdes Voufo[2]

[1] Programming and Software Engineering, University of Regensburg, Regensburg, Germany
judith.michael@ur.de
[2] Software Engineering, RWTH Aachen University, Aachen, Germany
{rumpe,stachon,stueber}@se-rwth.de, valdes.voufo@rwth-aachen.de
https://www.se-rwth.de/

Abstract. Reference process models are normative and prescriptive models that encapsulate best practices and standards for reuse and are thus essential for ensuring quality and consistency in concrete processes. Conformance checks are required to verify the adherence between a concrete process model of a system and its reference model.

This paper explores automated conformance checks using causal dependency analysis of tasks and events. Current methods primarily focus on verifying execution traces, lacking the necessary expressiveness and automation for semantic model-to-model comparison. Our approach is integrated into a broader semantic framework for defining reference model conformance. We present an algorithm for conformance checking, evaluate its implementation, and discuss its strengths and limitations. Our research offers a tool-assisted automated solution that enhances accuracy and flexibility in process model conformance verification.

Keywords: Reference Models · Process Models · Denotational Semantics · Conformance Checking · BPMN · Model-Driven Engineering

1 Introduction

Process models are crucial in various domains, providing structured representations of workflows. They standardize processes [5,63], optimize performance [36], and ensure compliance with industry standards [32,39]. As organizations increasingly rely on these models [19], robust mechanisms for verifying adherence of concrete implementations to reference models are essential.

Reference models are normative and prescriptive models that serve as benchmarks and encapsulate standardized processes and best practices [20,29]. However, their usage remains mostly informal, since formal conformance verification

J. Mendling et al. (Eds.): Wil van der Aalst Festschrift, LNCS 16480, pp. 525–550, 2026.
https://doi.org/10.1007/978-3-032-17618-9_35

methods for effective model-to-model comparisons are lacking. Current conformance checking approaches focus primarily on the alignment of execution logs [1,4,9,18,51,53,54,58,59,62] with process models, overlooking the often critical preservation of semantic properties from reference to concrete models.

Conformance checks [34] are vital to ensure that concrete process models align with reference models, facilitating adherence to best practices and timely identification of deviations. This alignment is essential for regulatory compliance and operational efficiency. However, the complexity and variability of real-world processes complicate conformance checking, as frequent changes in process structures challenge validation efforts. Existing methods of semantic model-to-model comparison often struggle with limited expressiveness and scalability, impeding effective validation in diverse environments [31,41,42,56,60].

This paper is a revised and extended version of an existing arXiv-preprint [61]. It introduces a novel algorithmic approach for tool-assisted model-to-model conformance checking, leveraging causal relations analysis to improve process verification automation. Our goal is to provide a more accurate and flexible solution for verifying the conformance of complex process structures to reference models.

Objective: We aim to develop a method that systematically analyzes causal dependencies in reference process models and compares them to the causal relation between corresponding elements in concrete models, advancing the state-of-the-art in process model verification.

Contributions:

- A semantic framework for reference process models and conformance
- An abstract description of a conformance checking algorithm
- A publicly available Java implementation for conformance checking
- An evaluation of the tool's validity and performance using constructed models
- A discussion of the capabilities and limitations of or approach

Please note that our perspective on this topic is coming from the software engineering community, more concretely, the model-driven engineering community. Thus, we might use terms different from those of the process mining community. However, our aim is to bridge research from these communities.

Structure: In Sect. 2, we present a motivating example that illustrates key concepts. Section 3 discusses related work in the field. We introduce our conformance checking algorithm in Sect. 4, addressing its complexity, soundness, and completeness. Section 5 outlines the implementation and provides an evaluation of the tool. In Sect. 6, we discuss precision, limitations, and potential threats to validity, concluding with a summary of findings and future work directions.

2 Motivating Example

Consider the process model displayed in Fig. 1, a reference process for scientific writing. After starting the process, the first task is `Research`. After `Research`

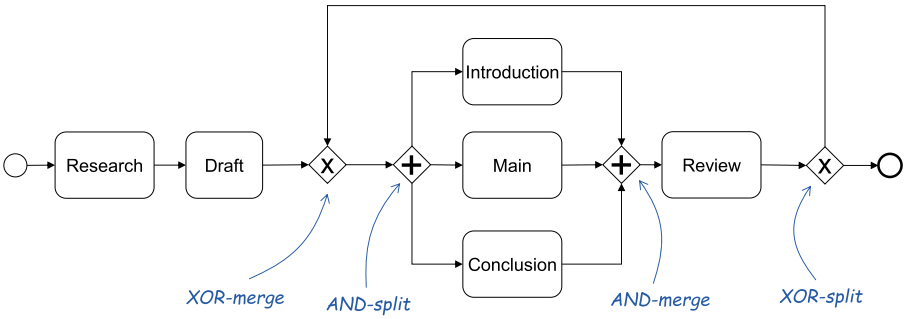

Fig. 1. Reference process model for scientific writing.

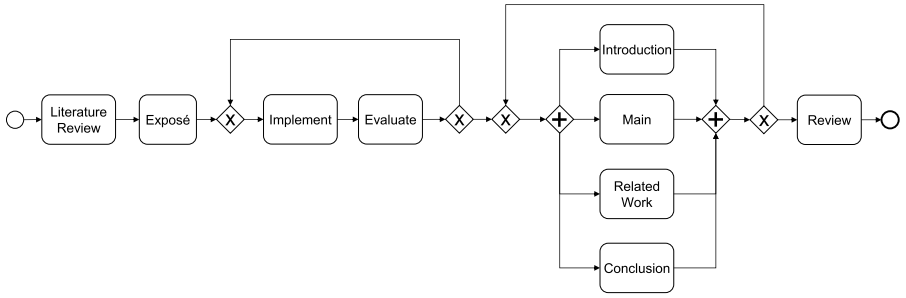

Fig. 2. Concrete process model for writing a thesis; the `Review` task is incorrectly placed after the final loop.

is completed, the next task is to write an initial `Draft`. Following that `Introduction`, `Main`, and `Conclusion` have to be completed. These three can be worked on in parallel. Next up is the task `Review`, after which the process either ends or the tasks `Introduction`, `Main`, and `Conclusion` are repeated.

Based on this reference model, we want to develop a more refined version of the model for the concrete case of writing a thesis. This model is displayed in Fig. 2: The thesis starts with a `Literature Review` instead of `Research`, followed by writing an `Exposé` instead of a `Draft`. Afterwards, the tasks `Implement` and then `Evaluation` have to be completed. Both may have to be repeated. Following that `Introduction`, `Main`, `Conclusion`, as well as `Related Work` have to be written. After a `Review` of the work, these tasks may have to be repeated, as well. Otherwise, the process ends.

This is the intended specification of the process. However, the modeler made a mistake and placed the `Review` task after the final loop. As a consequence, the execution behavior of the model does not *conform* to the reference model, *i.e.,* its semantics were not properly preserved.

A mistake such as this becomes harder to discover the larger the model grows, increasing the chances that the model will be implemented and executed when the corresponding software system is deployed. The severity of the impact such a

mistake has can vary, but if compliance to the reference model is legally required, it is best to avoid this situation in the first place by utilizing tool-assisted model-to-model conformance checks.

3 Background and Related Work

We outline necessary background information and discuss related work.

Process Modeling with BPMN

Business Process Model and Notation (BPMN) [50] is an internationally recognized standard developed by the Object Management Group (OMG). Its primary aim is to furnish a notation for business process design that is easily comprehensible to all stakeholders across various levels of the development process, from initial drafts to implementation, and extending to the management and monitoring of these processes. BPMN serves as a synthesis of best practices from the business modeling community, delineating the notation and semantics for collaboration, process, and choreography diagrams.

In this paper, we focus on the process design aspects of BPMN, specifically addressing the order and interdependencies of task executions within a business process. Consequently, we restrict our analysis to a subset of the BPMN syntax and features. Our focus encompasses process definitions that involve *events*, *tasks*, *gateways*, and *sequence flows* that interconnect these elements. For our purposes, we categorize gateways into two types: *split* and *merge*. Each gateway can further be classified as an `AND` gateway, an `XOR` gateway, or an `OR` gateway.

According to the BPMN standard, the following execution semantics apply to gateways: All sequence flows following an `AND`-split gateway are executed in parallel. In contrast, after an `XOR`-split gateway, only one of the subsequent sequence flows is executed. `OR`-split gateways permit the parallel execution of multiple subsequent sequence flows. An `AND`-merge waits for the completion of all preceding sequence flows, while an `XOR`-merge requires the completion of just one preceding sequence flow. Notably, other active flows are not terminated and may continue to pass through the gateway. Lastly, an `OR`-merge awaits the completion of all active preceding sequence flows.

Conformance to a Reference Model

In the context of process mining, the terms *conformance checking* or *conformance testing* usually refer to a comparison of actual process executions recorded in event logs with the behavior specified by a process model, *i.e.,* the intended or required procedure [1,4,9,18,51,53,54,58,59,62]. We aim to extend this notion of conformance to a semantic model-to-model comparison between a reference model and a concrete model. Now, the focus shifts from determining whether a process execution trace represents a legal instance of a model to verifying whether all legal executions of the concrete model conform to the reference model.

A reference model is an abstract model within a given modeling language that captures domain concepts and their domain-specific relations. It specifies properties that must be maintained for any conformant concrete model, *e.g.,* which model elements may or must exist, and how they interrelate [33]. Thus, from a model-driven engineering perspective, a reference model presents a template-solution for a class of concrete problems, or in the case of reference process models, a family of business processes.

An alternative approach to this notion of a reference model is the concept of configurable process models, which capture multiple variants of a process model in a consolidated manner [37,57].

Conformance to a reference model is crucial, especially in cases where the reference model encodes compliance constraints. To formally specify conformance, a conformance relation is used. A robust conformance relation must guarantee that all relevant semantic properties are preserved by conformant concrete models [34]. Specifically, a conformant concrete model must contain appropriate *incarnations* of relevant elements from the reference model. These incarnations are concrete model elements that correspond to reference elements while preserving their interrelations in the concrete model.

To ensure this formally, we require semantic refinement of the reference model by the concrete model in the context of incarnation. This process is crucial to maintain the intended semantics of the reference model and ensure integrity in the relationships among model elements.

Incarnations are specified via an incarnation mapping—a formal specification that defines how elements in the reference model correspond to elements in the concrete model—or automatically derived by a conformance checking algorithm. Conformance checking algorithms implement these conformance relations. Such algorithms have been developed for various modeling paradigms, including class diagrams, feature models, and statecharts [34]. They evaluate how well a concrete model adheres to its reference model and determine conformance violations that occur in the concrete model.

The incarnation mappings for these algorithms were either specified using stereotypes [21,28], which annotate model elements with additional semantics, or through custom mapping languages that provide a flexible way to define relationships between model elements. In cases where explicit mappings are incomplete or ambiguous, name-equality serves as a fall-back option, allowing the algorithm to infer correspondences based on the similarity of element names.

In this paper, we present a novel conformance checking algorithm specifically designed for process models. Our approach leverages stereotypes for encoding the incarnation mapping while also employing name-equality as a fall-back option to enhance the robustness of the conformance checking process.

With our conformance checking approach, we aim to address process compliance at design time. Runtime compliance, which is ensured through compliance monitoring [55], is beyond the scope of this paper.

Semantic Differencing

Conformance checking requires a formal notion of semantics. We employ a denotational semantics definition $sem : M \rightarrow 2^D$ which assigns each syntactically correct model of a modeling language M a set of valid instances within a well-defined and comprehensible semantic domain D [26]. In this context, refinement is characterized as a subset relation between sets of instances; specifically, a model A is said to refine a model B if and only if $sem(A) \subseteq sem(B)$. For process models, valid instances correspond to process execution traces.

Denotational semantics is employed in a variety of semantic differencing operators for multiple modeling languages [10,11,15,16,30,31,38,40,42,56,60]. These operators compare two input models with respect to their legal instances and can be extended with incarnation mappings to allow for conformance checking [34].

For concrete models to meaningfully extend reference models, it is generally more appropriate to adopt an open-world assumption regarding model semantics for conformance relations, rather than a closed-world assumption. This perspective allows a concrete model to incorporate additional elements that do not have counterparts in the reference model while still maintaining conformance. In terms of process model semantics, this implies that the execution trace of the concrete model may include additional tasks and events, provided that the causal relations among tasks and events in the reference model are preserved by their incarnations. It is essential to note that we assume tasks and events are uniquely identified by names within a process model, ensuring that each appears only once.

One approach to formalizing the execution semantics of process models, particularly Business Process Model and Notation (BPMN), is to translate them into Petri nets [3,14]. For example, Medeiros et al. [45] utilize a Petri net representation to evaluate the precision of process models derived from event logs via a genetic algorithm. They compare the execution behavior of the derived model with that of the original process model used to build the synthetic event log. However, this comparison only considers the traces contained within the event log, which is insufficient to guarantee a refinement of the original model. In general, comparing Petri nets based on their execution traces is undecidable [27], so a simpler model would be preferable.

Existing semantic differencing operators for process models [31,40] convert activity diagrams into state machines, where the state space is represented by the power set of activities, and the transition function encodes all potential process execution steps, namely, transitions from one set of active tasks to another. In this context, [40] employs this translation to perform bisimulation of the models, while [31] utilizes language inclusion checking algorithms for finite word automata.

The translation to state machines significantly increases the state space compared to the original activity diagram, leading to scalability issues due to the power-set automaton construction necessary for capturing the semantics of concurrent activities. Although it should be noted that state space reduction is possible and alternative concurrency models, such as partially ordered multisets

(pomsets) and Mazurkiewicz traces [6,44,52] exist, which address this issue by utilizing partial orders and equivalence classes that maintain causal dependencies and independence. However, these models introduce additional mathematical complexity and require specialized algorithms for determining behavioral equivalences, such as concurrency-aware bisimulation.

Another complicating factor in using bisimulation for conformance checking is the handling of multiple and composite incarnations of reference tasks and events. Multiple incarnations occur when a single reference element is incarnated multiple times in the concrete model, while composite incarnations involve a single reference element being represented by a configuration multiple elements in the concrete model or vice versa. For example, in process models, multiple incarnations may appear as inclusive or exclusive alternatives of tasks or events, whereas composite incarnations require the parallel or sequential execution of these tasks and/or events.

The challenge lies in bisimulation's requirement for a one-to-one correspondence between states in the reference and concrete models. This requirement becomes problematic with multiple and composite incarnations, as they introduce complexity that cannot be easily reconciled with the strict nature of bisimulation. Additionally, due to our open-world assumption regarding process model semantics, extra tasks and events may be added to the concrete model, further complicating the conformance checking process. Unlike these additional elements, multiple and composite incarnations cannot simply be deleted or ignored in the concrete model, as they are integral to its structure and behavior.

Our approach circumvents the need for full bisimulation by focusing exclusively on local causal dependencies, accepting a trade-off in completeness. Initially, we construct two propositional formulas for each task and event in the reference model: one formula captures the causal dependencies to its direct predecessor tasks and events, while the other addresses the dependencies to its direct successors. We then verify whether these dependencies are maintained for the corresponding incarnations in the concrete model.

This approach is in principle similar to the compliance analysis based on behavioral profiles developed by Weidlich et al. [64]. A behavioral profile considers binary order relations between activities in a process log or model. It can be extended to a causal behavioral profile by adding a binary causal relation. These relations can be efficiently computed for sound process models without `OR` gateways as well as process logs. From there several metrics for compliance of a process log to a process model can be derived.

However, we believe that the causal behavioral profile is unsuited to serve as the semantic basis for our notion of reference process model conformance, as the addition of further alternatives to an already existing set of exclusive alternatives would not violate any of the existing relations, despite being contrary to our notion of refinement and causal dependency preservation.

This might not have been an issue if we were simply computing a compliance metric, but we are interested in identifying (potential) violations of conformance in a concrete model, so that they may be addressed by the modeler.

4 Conformance Checking Approach

We define the open-world semantics of a process model as the collection of process execution traces that uphold the local causal dependencies of tasks and events as specified by the process model. In this context, each instance of a task or event is permitted to occur only after a suitable configuration of its predecessor instances has taken place and before an appropriate configuration of its successor instances is realized. Furthermore, these configurations of predecessor and successor instances must also be maintained between instances of the same task or event. In other words, loops that disrupt the local dependency structure of tasks and events are explicitly disallowed.

Our conformance checking algorithm operates under the assumption that an incarnation mapping exists which relates each incarnation in the concrete process model to a corresponding reference task or event in the reference model. Notably, each reference element may have multiple incarnations. We analyze each reference element and its incarnations individually, scrutinizing its local causal dependencies with respect to both its predecessor and successor tasks and events as encoded in the graph structure and gateways of the reference process model. This analysis is then juxtaposed with the causal dependencies of the corresponding incarnations in the concrete model.

This approach bears similarity to our conformance checking method for class diagrams as described in [34], where an incarnation is deemed conformant to its corresponding reference element if its properties and relationships with other elements are preserved. However, in the context of concrete process models, it is inadequate to limit our examination to neighboring tasks and events. For example, consider the following scenarios:

- **Insertion of New Tasks or Events:** A new task or event may be introduced between two incarnations of subsequent reference elements. This situation reflects a refinement in accordance with our open-world semantics, as it allows for the evolution of the process model without violating causal dependencies.
- **Sequential Execution of Parallel Tasks:** In the concrete model, the incarnations of two parallel tasks might need to be executed sequentially. This represents a refinement, since it alters the execution order while still preserving the underlying dependencies defined in the reference model.
- **Violation of Exclusive Alternatives:** The relationships defined by exclusive alternatives among reference tasks may be contravened in the concrete model if their incarnations are executed sequentially. In such cases, the concrete model fails to be a refinement of the reference model and thus does not conform to it.

Algorithm

The conformance checking algorithm is divided into two distinct phases. In the first phase, we identify the local causal dependencies of tasks and events within

the reference model and encode these dependencies into propositional logic formulas, one for all direct predecessors and another for all direct successors. In the second phase, we perform the conformance check on the concrete model by analyzing the causal relation of each incarnation to its predecessors and successors in the concrete model and subsequently comparing these relations to the local causal dependencies expressed in the corresponding formulas.

Phase 1: Construct the Formula In this phase, we compute the direct predecessors and successors of a task or event in the reference model and represent its interrelations as formulas in propositional logic, treating each task and each event as a Boolean variable. This process involves executing a depth-first search both forwards and backwards from the reference element.

During the forward search, we branch out at each split gateway, while in the backward search, we branch out at each merge gateway, continuing until we encounter the first task or event on each branch. Each gateway is interpreted as a corresponding logical operation applied to the sub-formulas derived from the branches. The pseudocode for the forward direction is presented in Algorithm 1.

Algorithm 1. A recursive algorithm for computing the successor formula of n

Require: x.suc are the predecessor nodes of x
  return SucForm(n.suc)
  function SucForm(x)
    if x is an `AND`-split gateway **then**
      return $AND(\{\text{SucForm}(s)\colon s \in x.\text{suc}\})$
    else if x is an `XOR`-split gateway **then**
      return $XOR(\{\text{SucForm}(s)\colon s \in x.\text{suc}\})$
    else if x is an `OR`-split gateway **then**
      return $OR(\{\text{SucForm}(s)\colon s \in x.\text{suc}\})$
    else if x is an event or task of the reference model **then**
      return x
    else
      return SucForm($x.suc$)
    end if
  end function

In the backward direction, we treat `XOR`-merge gateways identically to `OR`-merge gateways, reflecting the execution semantics of BPMN. Specifically, multiple preceding sequence flows can be concurrently active and reach the gateway. The corresponding pseudocode for this process can be found in Algorithm 2.

Example: Consider the task `Draft` in the reference process model from Fig. 1. Its only predecessor is `Research`, so the formula for the backward direction consists only of the Boolean variable of the same name. For successors, we find `Introduction`, `Main`, and `Conclusion` after an `AND`-split gateway, which means that the formula for the forward direction is the following:

`Introduction AND Main AND Conclusion`

Algorithm 2. A recursive algorithm for computing the predecessor formula of n

Require: x.pred are the successor nodes of x
  return PreForm(n.pred)
  function PreForm(x)
    if x is an `AND`-merge gateway **then**
      return $AND(\{\text{PreForm}(p) : p \in x.\text{pred}\})$
    else if x is an `XOR`- or `OR`-merge gateway **then**
      return $OR(\{\text{PreForm}(p) : p \in x.\text{pred}\})$
    else if x is an event or task of the reference model **then**
      return x
    else
      return PreForm($x.pred$)
    end if
  end function

If we consider the task `Review`, instead, we get an identical formula but for the backward direction. As for the forward direction, `Review` is assigned the formula:

`(Introduction AND Main AND Conclusion) XOR Done`

with `Done` being the end event.

Phase 2: Check Conformance After constructing the two formulas encoding the local causal dependency of the reference task or event to either its direct predecessors and its direct successors, we perform a quasi-simulation of the concrete model—both forward and backward—using breadth-first search, starting with the incarnation of the reference element. The forward search aims to determine whether the incarnations of the successors can be located in a configuration that satisfies the corresponding formula, while the backward search serves a similar purpose for the predecessors.

In the following, we focus solely on the forward direction, as the backward direction follows a largely analogous approach. A simplified version of our algorithm for the forward direction is presented as pseudocode in Algorithm 3. We initiate the process with the incarnation n and establish the initial branch b. Each branch comprises a set of visited nodes N, a set of active nodes A, and a result r. Additionally, we maintain the current execution trace, although this detail is omitted from the pseudocode for simplicity.

Algorithm 3. Simplified conformance checking algorithm for an incarnation n

Require: x.suc are the successor nodes of x and x.pred the predecessors

$\mathcal{B} \leftarrow \{(\emptyset, \{n\}, \text{not conformant})\}$
while $\exists b = (N, A, r) \in \mathcal{B}$ with $A \neq \emptyset$ **do**
  for all $b = (N, A, r) \in \mathcal{B}$ with $A \neq \emptyset$ **do**
    for all $x \in A$ **do**
      if x is an event, a task, an `AND`-split gateway, or an `XOR`- or `OR`-merge gateway **then**
        $A \leftarrow (A \backslash \{x\}) \cup (x.\text{suc} \backslash N)$
        $N \leftarrow N \cup x.\text{suc}$
        UpdateResult(b)
      end if
    end for
    if $\exists x \in A : x$ is an `XOR`- or `OR`-split gateway **then**
      if x is an `XOR` gateway **then**
        for all $s \in x.\text{suc}$ **do**
          $b_s \leftarrow (N \cup \{s\}, (A \backslash \{x\}) \cup (\{s\} \backslash N), r)$
          UpdateResult(b_s)
          $\mathcal{B} \leftarrow \mathcal{B} \cup \{b_s\}$
        end for
      else if x is an `OR` gateway **then**
        for all $S \subseteq x.\text{suc}$ with $S \neq \emptyset$ **do**
          $b_S \leftarrow (N \cup S, (A \backslash \{x\}) \cup (S \backslash N), r)$
          UpdateResult(b_s)
          $\mathcal{B} \leftarrow \mathcal{B} \cup \{b_S\}$
        end for
      end if
      $\mathcal{B} \leftarrow \mathcal{B} \backslash b$
    else if $A \neq \emptyset$ **then**
      while A contains only `AND`-gateways **do**
        if $\exists m \in A : m.\text{pred} \subseteq N$ **then**
          $x \leftarrow m$
        else
          $x \in A$
        end if
        $A \leftarrow (A \backslash \{x\}) \cup (x.\text{suc} \backslash N)$
        $N \leftarrow N \cup x.\text{suc}$
        UpdateResult(b)
      end while
    else if $n \notin N$ and N contains no end event **then**
      $\mathcal{B} \leftarrow \mathcal{B} \backslash b$
    end if
  end for
end while
return $\mathcal{B}$
function UpdatResult($b = (N, A, r)$)
  if $r =$ not conformant and N satisfies the formula **then**
    $r =$ conformant
  else if $r =$ conformant and N does not satisfies the formula **then**
    $r =$ unknown
  end if
end function

The algorithm proceeds iteratively until no branches with active nodes remain. In each iteration, for every event, task, AND-split gateway, and XOR- or OR-merge gateway present in A, we perform the following steps to progress:

1. Remove the current node from A.
2. Add all successor nodes that are not already included in N to A.
3. Update the result of the branch by checking whether the current set of tasks and events in N satisfies the encoded formula.

Due to the presence of exclusive alternatives, we may encounter situations where a branch previously marked as *conformant* no longer satisfies the formula, leading us to designate its status as *unknown*. This does not necessarily imply that the execution trace represented by this branch is non-conformant.

When encountering an XOR-split gateway in A, a new branch is created for each successor node, updating the sets of visited and active nodes for each new branch, after which the current branch is deleted. Similarly, if an OR-split gateway is encountered, a new branch is created for each subset of successor nodes before deleting the current branch. Conversely, if the set of active nodes A contains only AND-merge gateways, we can progress through one of these gateways if all its predecessor nodes are included in N; otherwise, we will advance through another available AND-merge gateway.

If no active nodes remain and we have not yet reached an end event or returned to nn, we delete the branch. Once all branches with active nodes have been exhausted, we return the set of branches that were not deleted to examine their results. If any non-conformant branch exists, we conclude that the incarnation is *not conformant*, returning the execution trace as a diff witness. Conversely, if a branch with the status *unknown* exists, the conformance status of the incarnation is designated as *unknown*, and we return the trace for potential manual verification. Lastly, if all remaining branches are conformant, we classify the incarnation as *conformant*.

Example: Going back to our previous example, we now consider the concrete process model from Fig. 2, where we start the forward search for the task `Review`. We branch out in our search because of the subsequent XOR-split gateway. One branch terminates in the next step as we reach the end event `Done`. Having visited `Done`, the branch satisfies the successor formula:

`(Introduction AND Main AND Conclusion) XOR Done`

The other branch finds an AND-split gateway after the loop and adds the tasks `Introduction`, `Main`, `Related Work`, and `Conclusion` to its list of visited nodes. In the next step, we reach the starting point `Review` and terminate the search in this branch. This branch also satisfies the successor formula, having visited:

`[Introduction, Main, Related Work, Conclusion, Review]`

Since all branches satisfy the formula, the task `Review` is conformant with respect to its successors. However, this is not the case with regard to its predecessors.

If we backtrack, we immediately encounter an `XOR`-split and branch out. One of the branches will now, before terminating, visit:

`[Related Work, Evaluate, Implement, Exposé, Literature Review, Start]`

This does not satisfy the predecessor formula:

`Introduction AND Main AND Conclusion`

As such, the task `Review` is not conformant.

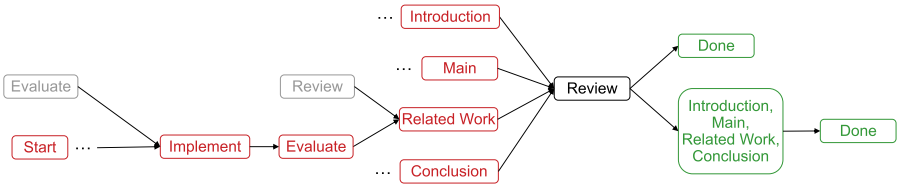

Fig. 3. Abbreviated search tree for forward and backward search for the task `Review` in the concrete model in Fig. 2 with satisfying branches in green, non-satisfying in red, and branches deleted due to loops/idleness in gray.

The abbreviated search tree for both the forward and backward search is illustrated in Fig. 3. The nodes contain the tasks and events visited in that step and are colored green if the formula is satisfied in this step, red if not, and gray if an already visited element was visited once again and the branch will be ignored.

Complexity: If no inclusive `OR` gateways are utilized, both parts of the algorithm can be executed in polynomial time. Specifically, we first employ depth-first search to construct the formula, followed by breadth-first search for the conformance checking. However, the complexity increases exponentially with the number of inclusive decision branches. Notably, this represents an improvement over previous approaches that relied on power-set automaton construction [31,40], as our method does not require interleaving concurrent tasks and events.

Soundness: A conformance relation must ensure semantic refinement in the context of incarnations. For process models, this means that every execution trace of the concrete model must align with the reference model. To enable process extension in the concrete case, we operate under an open-world assumption. As such, the relative order of elements in a trace must correspond to the causal dependencies of reference elements.

Our algorithm checks for the preservation of causal dependencies under incarnation, which serves as a sufficient condition for semantic refinement under the open-world assumption: At any point during the concrete process's execution, it guarantees for each active incarnation that all predecessor incarnations can ultimately be identified through backtracking, and a suitable configuration of

successor incarnations will eventually be established if the execution is continued. Consequently, the resulting execution trace must align with the reference model.

Completeness: The reduced complexity of our approach compared to previous semantic differencing methods for process models [31,41] comes at a cost. In certain cases, a branch may reach a configuration that violates the formula concerning an `XOR`-constraint after having satisfied the formula in a prior step. This circumstance does not necessarily indicate that the model is non-conformant, as illustrated by the process model depicted in Fig. 4.

For instance, when checking the conformance of this model against itself, the algorithm first derives the formula `B XOR C` for the successors of `A`. During the conformance check of `A` as an incarnation of itself, the algorithm will branch in the first step due to the `XOR`-split and will successfully identify satisfying configurations `B` and `C`, respectively. However, in the subsequent step, the configuration `[B,C]` is encountered, which no longer satisfies the formula. Consequently, the branch is unable to reach a satisfying configuration thereafter.

As a result, the algorithm indicates that it cannot ascertain whether `A` is conformant, outputting the configuration `[B,C]`. Nevertheless, this task sequence `[B,C]` can be extended to form a legal run `[start,A,B,C,end]`, as the reference model is identical to the concrete model. This allows for manual verification, confirming that the model is indeed conformant.

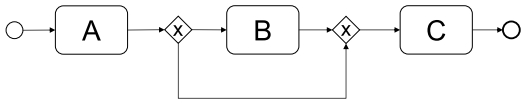

Fig. 4. Example of a process model with a skippable task.

5 Implementation and Tool

For our implementation, we use a textual variant of BPMN [17] developed with the `MontiCore`[1] language workbench [22,28]. To encode the incarnation mapping, we have extended the grammar of the language to allow the annotation of tasks and events with stereotypes [21,28]. Similar to our previous conformance checking approach of class diagrams [34], incarnations are identified via stereotypes that specify the name of the incarnation mapping as well as the name of the corresponding reference element. If for a given reference element no incarnation is specified, a concrete element of the same name is considered as incarnation.

Listing 1.1 shows a process model in the textual BPMN syntax that describes a sequential writing process. The tasks `Concept` and `Implementation` are each

[1] https://monticore.github.io/monticore/.

annotated with a stereotype indicating that both tasks are mapped via the incarnation mapping `ref` to a task in a reference model with the name `Main`. This process does in fact conform to our reference model for scientific writing displayed in Fig. 1, since the order of tasks has simply been sequentialized.

The conformance-checking tool has been integrated into the `BPMN` language project of the `MontiCore` language family and is publicly available on GitHub[2]. After building the project the BPMN conformance check can be executed via the `BPMN.jar`. The tool takes as input a path to the reference model, specified via the option $-r$ and a path to the concrete model, specified via the option $-c$. Both must be in the form of a `.wfm`-file containing the textual specification. Finally, the name of the incarnation mapping used in the concrete model is specified via the option $-m$. An example command as well as examples of potential outputs are presented in the Appendix.

```
process SequentialWriting {
  event start Start;
  event end Done;

  task Research;
  task Draft;
  task Introduction;
  <<ref="Main">> task Concept;
  <<ref="Main">> task Implementation;
  task Conclusion;
  task Review;

  Start -> Research -> Draft -> Introduction -> Concept
    -> Implementation -> Conclusion -> Review -> Done;
}
```

Listing 1.1. Example – BPMN in textual syntax.

Evaluation

We evaluated both the validity of our approach and the output and performance of our implementation using constructed process models. The tests for the case study and performance analysis—including the algorithm for constructing the performance test model—can be found in the BPMN project at [49] with the corresponding model-files for the case studies at [48].

Validation: In order to validate our approach and evaluate the output of our implementation, we constructed a small case study, in which we consider conformant and non-conformant extensions and modifications of reference models using the reference process model for scientific writing displayed in Fig. 1 as our initial model. Noticeably, we have included ten changes that we consider conformant, and ten that are non-conformant.

[2] https://github.com/MontiCore/bpmn.

We consider the following modifications refining, *i.e.,* the resulting models should conform to the original model:

1. sequentializing parallel tasks
2. removing a loop
3. adding new tasks
4. removing alternatives
5. parallelizing inclusive alternatives
6. transforming inclusive into exclusive alternatives
7. incarnating a task multiple times in parallel
8. incarnating a task multiple times in sequence
9. incarnating a task multiple times as inclusive alternatives
10. incarnating a task multiple times as exclusive alternatives

We consider the following modifications non-refining, *i.e.,* the resulting models should not conform to the original model:

1. switching the order of tasks
2. removing or not incarnating a task
3. incarnating a task at a correct and incorrect position
4. transforming an `XOR`-split into an `AND`-split
5. transforming an `AND`-split into an `XOR`-split
6. transforming an `AND`-merge into an `XOR`-merge
7. transforming an `AND`-split into an `OR`-split
8. parallelizing exclusive alternatives
9. turning exclusive alternatives into inclusive alternatives
10. sequentializing exclusive alternatives

We perform a conformance check to compare the modified model to the original and verify the results. In all cases the tool identifies the non-conformant nodes and outputs a corresponding diff witness in the form of a run or backtrack sequence.

Performance and Scalability: For our performance test, we designed the process model shown in Fig. 5. It contains ten tasks, two branches of parallel activities, two branches with exclusive alternatives, and a loop. We then implemented an algorithm that automatically constructs duplicates of this model and concatenates them. For our approach, it is also necessary to relabel the tasks so that their names remain unique. This is done by adding indices to the task names.

Each concatenation increases the size of our model by ten tasks. For each concatenated model, we also produced a duplicate with a small modification: the task $G1$ is moved to the other branch right after $H1$. As a result, the modified model does not conform to the original.

We used our conformance checking tool to compare ten pairs of models, starting with the base model in Fig. 5 and iteratively increasing the size by ten tasks each step through concatenation. We validated the results of the conformance

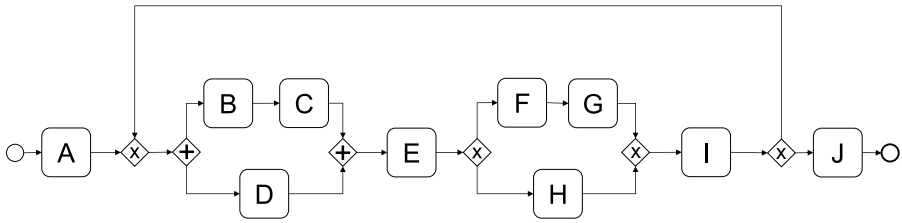

Fig. 5. Process model used as the basis for the performance test.

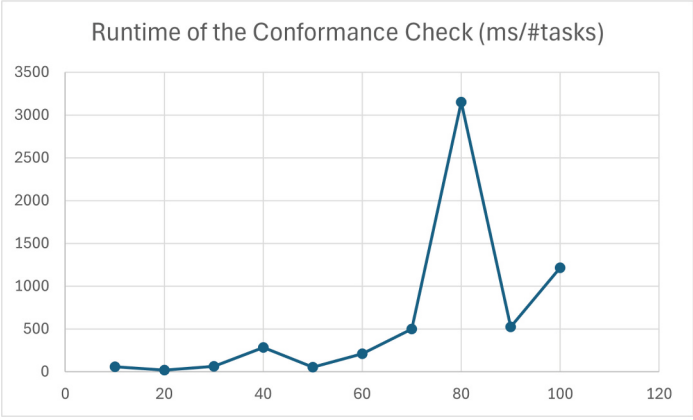

Fig. 6. Runtime of the conformance checking procedure in milliseconds (y-axis) compared to the size of the model/number of tasks (x-axis).

check and measured the performance of our tool in milliseconds. All experiments were performed on an 11th Gen Intel Core i7-1185G7 CPU, 3.0 GHz, with 32 GB RAM, running Windows 11. The results are displayed in Fig. 6.

The runtime of our conformance check remains under half a second until we reach a size of 70 tasks in our model. We see a sudden spike in the runtime at 80 tasks, where the conformance check takes more than 3 s to terminate.

Despite this sudden spike in runtime, our approach seems, at least at first glance, to scale significantly better than our original semantic differencing operator for activity diagrams [43].

Moreover, further optimization is possible through parallelization. For example, each incarnation can be independently checked, and the branches produced by our algorithm can be parallelized as well.

6 Discussion

In this paper, we explored conformance checking between concrete and reference process models, developing an approach that preserves causal dependencies

among tasks and events in their incarnations. This section addresses aspects of uncertainty, limitations, and potential threats to validity within our approach.

Uncertainty: The algorithm we developed ensures that during the execution of a conformant concrete model, tasks and events from the reference model can only occur if their predecessor incarnations have been executed, and suitable configurations of successor incarnations follow. This design guarantees that the execution trace remains aligned with the reference model in the context of incarnation under the open-world assumption.

When the algorithm identifies a non-conformant incarnation, it reveals a path that violates local dependencies as specified by the reference model. If the incarnation is reachable and a path to an end event exists, this path can be expanded into a *diff witness*, representing a process execution trace not permitted by the reference model. Uncertainty arises only from incarnations with undetermined conformance status, labeled as *unknown*. This occurs when a satisfying configuration is followed by a non-satisfying one in the same search branch, often due to exclusive alternatives among successors.

Unlike previous model-to-model conformance checking approaches that exhibit uncertainty regarding semantic refinement [34], our method cannot categorically label these cases as non-conformant, as reflexivity is essential in a conformance relation (cf. Fig. 4). Thus, we acknowledge our approach as somewhat incomplete, requiring manual review of these ambiguous instances. Fortunately, our algorithm provides the name of the individual incarnation and the execution trace flagged as potentially non-conformant, aiding this review process.

Limitations: A significant limitation of our current approach is that it only considers a subset of BPMN language features [50]. Specifically, we focus on basic tasks, start and end events, and the logical gateways `XOR`, `OR`, and `AND`, which together represent a rudimentary foundation for process modeling. However, certain features, such as lanes, are not integrated into our current trace-based semantics definition and may therefore be overlooked.

Future work should explore additional BPMN features and their implications for process semantics. In particular, executable formal expressions play a crucial role in defining flow and loop conditions, as well as event triggers. Our textual process model syntax supports such expressions, and we could extend our approach to include conformance checks for them by utilizing SMT-solving to verify the refinement of individual reference expressions by their concrete counterparts.

Moreover, sub-processes and call activities could be managed through a hierarchical conformance check, while data and message flows represent another critical aspect for consideration. In our BPMN variant's implementation, we can define usable data types using class diagrams [24,25], enabling us to leverage our existing conformance checks [34]. If behaviors are specified through operation constraints in OCL [13,60], operations declared in a class diagram and referenced in a task of the process model could also be subjected to conformance verification.

To enhance our approach further, it is essential to enable a more precise encoding of incarnation mappings—not only to support additional features but

also to facilitate more complex incarnations of currently supported features, such as tasks. The existing mapping mechanism, which relies on stereotypes, lacks sufficient expressiveness; for example, it does not allow a task in the reference model to be represented as a sequence of tasks. To overcome these limitations, we propose developing a custom mapping language to enhance the expressiveness and versatility of our incarnation mappings.

In addition, we will need to address compatibility and integration with existing tools in the business process management and process mining communities. Luckily, the conformance check itself is not directly tied to our textual BPMN variant and we have already begun work on an XML-translation.

Threats to Validity: Our current interpretation of open-world semantics views any extension of the original model as a refinement that preserves local causal dependencies between tasks and events, ensuring that our approach is effectively correct by construction. However, this definition may not apply universally; alternative interpretations could be more suitable in certain contexts.

For instance, we disallow adding loops that involve tasks and events present in the model by requiring suitable configurations of direct successors and predecessors between two instances of the same task or event within an execution trace. However, in some scenarios, these loops may represent necessary refinement steps, and it might suffice for the predecessor and successor configurations to occur just once, before and after all instances of the task or event within the trace.

To further validate our approach, we plan to enhance our evaluation by identifying relevant example cases from business, industry, and scientific literature for a comprehensive analysis.

7 Conclusion

This paper introduces an innovative approach to reference process model conformance checking, providing an algorithmic solution that leverages causal dependency analysis. Our motivation was to improve the expressiveness and automation of conformance checks, enabling more precise verification of complex process structures. We developed a method that systematically examines the causal interdependency of reference model elements and compares it with the causal relation between corresponding elements in concrete models, thus providing a comprehensive framework for conformance verification.

Our contributions include establishing a semantic concept for reference process models and conformance, providing an abstract description of the conformance checking algorithm, releasing a publicly accessible Java implementation, and evaluating the tool with multiple examples. These efforts advance the field by offering a robust methodology and toolset for improving conformance checks.

Future work will involve conducting industry case studies to validate our approach in real-world scenarios, extending our BPMN feature support to cover a broader range of elements and language variants [12,23], and integrating our algorithm into existing toolsets, *e.g.,* via plugin. Additionally, we intend to make

further improvements to our tool's performance by leveraging parallelization. These efforts aim to enhance the practical applicability and robustness of our method across diverse industrial contexts.

Acknowledgments. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 250902306. All authors contributed equally.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Traces of Thought: In Honor of Wil van der Aalst

In the ever-evolving landscape of computer science, there are figures who do more than advance a field. They shape it, define it, and inspire generations of scientists through their vision and perseverance. Among these rare individuals stands Professor Wil van der Aalst, a pioneer whose contributions to process and workflow modeling, and process mining have left a lasting intellectual and institutional legacy.

As a result, Wil has left an indelible mark on colleagues, students, the broader academic community, but also industry, where even a Decacorn has meanwhile created to a large extent on his ideas. As colleagues privileged to witness his impact firsthand within RWTH Aachen University, we are deeply impressed, by his career and the impact he produced.

Wil van der Aalst's scientific journey is nothing short of monumental. With over 1,000 publications, countless keynote addresses, and the creation of process mining as a field in its own right, he has shown that innovation requires both precision and imagination. His seminal works—from the introduction of the α-algorithm [2] (together with colleagues Weijters and Maruster) to the foundational texts on specifying, identifying or mining workflow nets (again with colleagues) have become cornerstones for students and researchers alike.

But beyond the theorems and algorithms, what sets Wil apart is his unwavering commitment to relevance: bridging the gap between theory and application, ensuring that process science remains grounded in real-world systems and societal needs. He reminds us that research should not only be rigorous, but purposeful. He lives by the advice he is giving PhD students (see for his interview in [46]): Not to follow the crowd, do something original while still being able to explain what you do in the real world.

Wil's intellectual influence extends far beyond his papers. He has mentored many of doctoral students, many of whom now lead their own research groups across the world. He has built institutions—both formal and informal—where inquiry flourishes. Finally, his leadership at RWTH Aachen's Process and Data Science (PADS) Group has turned it into a global nucleus for process mining research. It was a pleasure to work with him and his great team for several years in the Cluster of Excellence Internet of Production on topics such as Models-in-the-Moddle [35], digital shadows [47], process prediction with digital twins [8],

and further topics that Computer Science can contribute to the digital transformation of production [7].

In conference halls, classrooms, and informal lunch discussions, Wil has continuously cultivated a spirit of generosity and collaboration. He listens as attentively as he lectures, and he engages as rigorously with a first-year student as with a senior peer. This rare humility in a scholar of his stature only deepens the respect he commands.

To Wil: Your intellectual rigor, your boundless curiosity, and your commitment to building a scholarly community are a guiding light. This paper is only a small gesture to honor a career whose influence cannot be quantified. Thank you for showing us how to think deeply, collaborate generously, and lead with integrity.

Bernhard Rumpe, Max Stachon, Judith Michael
(now or formerly at RWTH Aachen University)

Appendix

```
java -jar BPMN.jar                                    \
          -i Concrete.wfm -ref Reference.wfm          \
          -m "ref"
```

Listing 1.2. Example command for executing the BPMN conformance checker.

```
Checking Conformance of [Concrete:AntiPattern] to
[Reference:PaperAuthoring]

--- Final Result of Conformance Checking ---
The following nodes do not conform: [Main, Introduction,
Conclusion]

-------- Explanations --------:

Result: Node [MotivatingExample:Main] does not conform to
Node [PaperAuthoring:Main]
Counter example: The following run [Conclusion,
    Introduction,
Main] is possible in [MotivatingExample] but not in
[PaperAuthoring].

Result: Node [MotivatingExample:Introduction] does not
conform to Node [PaperAuthoring:Introduction]
Counter example: The following run [Conclusion,
    Introduction,
Main] is possible in [MotivatingExample] but not in
[PaperAuthoring].
```

```
21
22 Result: Node [MotivatingExample:Conclusion] does not
       conform
23 to Node [PaperAuthoring:Conclusion]
24 Counter example: The following run [Conclusion,
       Introduction,
25 Main] is possible in [MotivatingExample] but not in
26 [PaperAuthoring].
```

Listing 1.3. Example output in case of non-conformance, using the concrete model from fig. 2 and reference model from fig. 1.

```
1 Checking Conformance of [Concrete:Sequential] to
2 [Reference:PaperAuthoring]
3
4 --- Final Result of Conformance Checking ---
5 --- All nodes conform to their reference ---
```

Listing 1.4. Example output in case of conformance, using the concrete model from listing 1.1 and reference model from fig. 1.

```
1 Checking Conformance of [Concrete:Skip] to [Reference:
      Skip]
2
3 --- Final Result of Conformance Checking ---
4 The status of the following nodes is unknown: [A]
5
6 -------- Explanations --------:
7
8 Result: Node [Skip:A] may not conform to Node [Skip:A]
9 Counter example: The following run [B, C, Done] is
      possible
10 in [Skip] but may not be possible in [Skip].
```

Listing 1.5. Example output in case of potential non-conformance, using the model in fig. 4 as both concrete and reference model.

References

1. Van der Aalst, W., Adriansyah, A., Van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisc. Rev. Data Min. Knowl. Discov. **2**(2), 182–192 (2012)
2. Van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
3. Van der Aalst, W.M.: The application of petri nets to workflow management. J. Circuits Syst. Comput. **8**(01), 21–66 (1998)

4. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.: Conformance checking using cost-based fitness analysis. In: 2011 IEEE 15th International Enterprise Distributed Object Computing Conference, pp. 55–64. IEEE (2011)
5. Allweyer, T.: BPMN 2.0: introduction to the standard for business process modeling. BoD–Books on Demand (2016)
6. Bloom, B., Kwiatkowska, M.: Trade-offs in true concurrency: pomsets and mazurkiewicz traces. In: Brookes, S., Main, M., Melton, A., Mislove, M., Schmidt, D. (eds.) MFPS 1991. LNCS, vol. 598, pp. 350–375. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55511-0_18
7. Brauner, P., et al.: A computer science perspective on digital transformation in production. J. ACM Trans. Internet Things **3**, 1–32 (2022). https://doi.org/10.1145/3502265
8. Brockhoff, T., et al.: Process prediction with digital twins. In: International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 182–187. ACM/IEEE (2021)
9. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016). https://doi.org/10.1016/j.eswa.2016.08.040
10. Butting, A., Kautz, O., Rumpe, B., Wortmann, A.: Semantic differencing for message-driven component & connector architectures. In: International Conference on Software Architecture (ICSA 2017), pp. 145–154. IEEE (2017)
11. Butting, A., Kautz, O., Rumpe, B., Wortmann, A.: Continuously analyzing finite, message-driven, time-synchronous component & connector systems during architecture evolution. J. Syst. Softw. (JSS) **149**, 437–461 (2019). https://doi.org/10.1016/j.jss.2018.12.016
12. Cengarle, M.V., Grönniger, H., Rumpe, B.: Variability within modeling language definitions. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 670–684. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04425-0_54
13. Cook, S., Kleppe, A., Mitchell, R., Rumpe, B., Warmer, J., Wills, A.: The Amsterdam manifesto on OCL. In: Clark, T., Warmer, J. (eds.) Object Modeling with the OCL. LNCS, vol. 2263, pp. 115–149. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45669-4_7
14. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008)
15. Drave, I., Eikermann, R., Kautz, O., Rumpe, B.: Semantic differencing of statecharts for object-oriented systems. In: Hammoudi, S., Pires, L.F., Selić, B. (eds.) 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019), pp. 274–282. SciTePress (2019)
16. Drave, I., Kautz, O., Michael, J., Rumpe, B.: Semantic evolution analysis of feature models. In: Berger, T., et al. (eds.) International Systems and Software Product Line Conference (SPLC 2019), pp. 245–255. ACM (2019)
17. Drave, I., Michael, J., Müller, E., Rumpe, B., Varga, S.: Model-driven engineering of process-aware information systems. Springer Nature Comput. Sci. J. **3** (2022)
18. Dunzer, S., Stierle, M., Matzner, M., Baier, S.: Conformance checking: a state-of-the-art literature review. In: 11th International Conference on Subject-Oriented Business Process Management, S-BPM ONE 2019. ACM, New York (2019). https://doi.org/10.1145/3329007.3329014
19. Fernandes, J., Reis, J., Melão, N., Teixeira, L., Amorim, M.: The role of industry 4.0 and BPMN in the arise of condition-based and predictive maintenance: a case study in the automotive industry. Appl. Sci. **11**(8), 3438 (2021)

20. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Prentice Hall (1997)
21. Gogolla, M., Henderson-Sellers, B.: Analysis of UML Stereotypes within the UML Metamodel, pp. 84–99. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45800-x_8
22. Grönniger, H., Krahn, H., Rumpe, B., Schindler, M., Völkel, S.: MontiCore 1.0: Ein Framework zur Erstellung und Verarbeitung domänspezifischer Sprachen. Informatik-Bericht 2006-04, CFG-Fakultät, TU Braunschweig (2006)
23. Grönniger, H., Rumpe, B.: Modeling language variability. In: Workshop on Modeling, Development and Verification of Adaptive Systems. (16th Monterey Workshop). Redmond, Microsoft Research (2010)
24. Haber, A., et al.: Composition of heterogeneous modeling languages. In: Desfray, P., Filipe, J., Hammoudi, S., Pires, L.F. (eds.) MODELSWARD 2015. CCIS, vol. 580, pp. 45–66. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27869-8_3
25. Haber, A., et al.: Integration of heterogeneous modeling languages via extensible and composable language components. In: Model-Driven Engineering and Software Development Conference (MODELSWARD 2015), pp. 19–31. SciTePress (2015)
26. Harel, D., Rumpe, B.: Meaningful modeling: what's the semantics of "semantics"? IEEE Comput. J. **37**(10), 64–72 (2004)
27. Hirshfeld, Y.: Petri nets and the equivalence problem. In: Börger, E., Gurevich, Y., Meinke, K. (eds.) CSL 1993. LNCS, vol. 832, pp. 165–174. Springer, Heidelberg (1994). https://doi.org/10.1007/BFb0049331
28. Hölldobler, K., Kautz, O., Rumpe, B.: MontiCore Language Workbench and Library Handbook: Edition 2021. Aachener Informatik-Berichte, Software Engineering, Band 48, Shaker Verlag (2021)
29. ITU-T: Information technology – Open Systems Interconnection – Basic Reference Model: The basic model. ITU-T X.200, Int. Telecommunication Union (1994)
30. Kautz, O.: Model Analyses Based on Semantic Differencing and Automatic Model Repair. Aachener Informatik-Berichte, Software Engineering, Band 46, Shaker Verlag (2021)
31. Kautz, O., Rumpe, B.: Semantic differencing of activity diagrams by a translation into finite automata. In: MODELS 2018. Workshop ME (2018)
32. Knuplesch, D., Reichert, M., Pryss, R., Fdhila, W., Rinderle-Ma, S.: Ensuring compliance of distributed and collaborative workflows. In: 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, pp. 133–142. IEEE (2013)
33. Konersmann, M., Michael, J., Rumpe, B.: Towards Reference Models with Conformance Relations for Structure, pp. 247–269. Logos Verlag Berlin (2024)
34. Konersmann, M., Rumpe, B., Stachon, M., Stüber, S., Voufo, V.: Towards a semantically useful definition of conformance with a reference model. J. Object Technol. (JOT) **23**(3), 1–14 (2024). https://doi.org/10.5381/jot.2024.23.3.a5
35. Koren, I., et al.: Navigating the data model divide in smart manufacturing: an empirical investigation for enhanced AI integration. In: van der Aa, H., Bork, D., Schmidt, R., Sturm, A. (eds.) Enterprise, Business-Process and Information Systems Modeling, pp. 275–290. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-61007-3_21
36. Kougka, G., Gounaris, A., Simitsis, A.: The many faces of data-centric workflow optimization: a survey. Int. J. Data Sci. Anal. **6**, 81–107 (2018). https://doi.org/10.1007/s41060-018-0107-0

37. La Rosa, M., Dumas, M., ter Hofstede, A.H., Mendling, J.: Configurable multi-perspective business process models. Inf. Syst. **36**(2), 313–340 (2011). https://doi.org/10.1016/j.is.2010.07.001. Special Issue: Semantic Integration of Data, Multimedia, and Services
38. Langer, P., Mayerhofer, T., Kappel, G.: Semantic model differencing utilizing behavioral semantics specifications. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E. (eds.) Model-Driven Engineering Languages and Systems, pp. 116–132. Springer, Cham (2014)
39. Lim, H.W., Kerschbaum, F., Wang, H.: Workflow signatures for business process compliance. IEEE Trans. Dependable Secure Comput. **9**(5), 756–769 (2012)
40. Maoz, S., Ringert, J.O., Rumpe, B.: ADDiff: semantic differencing for activity diagrams. In: Conference on Foundations of Software Engineering (ESEC/FSE 2011), pp. 179–189. ACM (2011)
41. Maoz, S., Ringert, J.O., Rumpe, B.: An Operational Semantics for Activity Diagrams using SMV. Technical Report AIB-2011-07, RWTH Aachen University (2011)
42. Maoz, S., Ringert, J.O., Rumpe, B.: CDDiff: semantic differencing for class diagrams. In: Mezini, M. (ed.) ECOOP 2011 - Object-Oriented Programming, pp. 230–254. Springer, Heidelberg (2011)
43. Maoz, S., Ringert, J.O., Rumpe, B.: Modal object diagrams. In: Mezini, M. (ed.) ECOOP 2011. LNCS, vol. 6813, pp. 281–305. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22655-7_14
44. Mazurkiewicz, A.: Trace theory. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) ACPN 1986. LNCS, vol. 255, pp. 278–324. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-17906-2_30
45. de Medeiros, A.K.A., Weijters, A.J., van der Aalst, W.M.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Disc. **14**(2), 245–304 (2007)
46. Michael, J., Bork, D., Wimmer, M., Mayr, H.C.: Quo Vadis modeling? Findings of a community survey, an ad-hoc bibliometric analysis, and expert interviews on data, process, and software modeling. J. Softw. Syst. Model. (SoSyM) **23**(1), 7–28 (2024). https://doi.org/10.1007/s10270-023-01128-y
47. Michael, J., et al.: A digital shadow reference model for worldwide production labs. In: Brecher, C., Schuh, G., van der Aalst, W., Jarke, M., Piller, F., Padberg, M. (eds.) Internet of Production: Fundamentals, Applications and Proceedings, pp. 1–28. Springer, Cham (2023). https://doi.org/10.1007/978-3-030-98062-7_3-2
48. Michael, J., Rumpe, B., Stachon, M., Stüber, S., Voufo, V.: Workflow conformance test resources (2025). WorkflowConformance/src/test/resources/de/monticore/bpmn/conformance. Accessed 23 Sept 2025
49. Michael, J., Rumpe, B., Stachon, M., Stüber, S., Voufo, V.: Workflow conformance tests (2025). WorkflowConformance/src/test/java/de/monticore/bpmn/-conformance. Accessed 23 Sept 2025
50. Business process model and notation (2014)
51. Pegoraro, M., Uysal, M.S., van der Aalst, W.M.: Conformance checking over uncertain event data. Inf. Syst. **102**, 101810 (2021)
52. Pratt, V.: Modeling concurrency with partial orders. Int. J. Parallel Prog. **15**(1), 33–71 (1986)
53. Rafiei, M., van der Aalst, W.M.P.: Mining roles from event logs while preserving privacy. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 676–689. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_54

54. Rafiei, M., Pourbafrani, M., van der Aalst, W.M.: Federated conformance checking. Inf. Syst. **131**, 102525 (2025)
55. Rinderle-Ma, S., Winter, K., Benzin, J.V.: Predictive compliance monitoring in process-aware information systems: state of the art, functionalities, research directions. Inf. Syst. **115**, 102210 (2023)
56. Ringert, J.O., Rumpe, B., Stachon, M.: On implementing open world semantic differencing for class diagrams. J. Object Technol. (JOT) **22**(2), 2:1–14 (2023). https://doi.org/10.5381/jot.2023.22.2.a11
57. Rosemann, M., Van der Aalst, W.M.: A configurable reference modelling language. Inf. Syst. **32**(1), 1–23 (2007)
58. Rozinat, A., van der Aalst, W.M.P.: Conformance testing: Measuring the fit and appropriateness of event logs and process models. In: Bussler, C.J., Haller, A. (eds.) Business Process Management Workshops, pp. 163–176. Springer, Heidelberg (2006)
59. Rozinat, A., Van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
60. Rumpe, B., Stachon, M., Stüber, S., Voufo, V.: Semantic difference analysis with invariant tracing for class diagrams extended by OCL. In: WS on Model Driven Engineering, Verification and Validation (MoDeVVa), MODELS Companion 2024. ACM (2024). https://doi.org/10.1145/3652620.3687818
61. Rumpe, B., Stachon, M., Stüber, S., Voufo, V.: Tool-assisted conformance checking to reference process models. arXiv (2025). https://doi.org/10.48550/arXiv.2508.00738
62. Schuster, D., Kolhof, G.J.: Scalable online conformance checking using incremental prefix-alignment computation. In: Hacid, H., et al. (eds.) ICSOC 2020. LNCS, vol. 12632, pp. 379–394. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-76352-7_36
63. Ungan, M.C.: Standardization through process documentation. Bus. Process. Manag. J. **12**(2), 135–148 (2006). https://doi.org/10.1108/14637150610657495
64. Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., Weske, M.: Process compliance analysis based on behavioural profiles. Inf. Syst. **36**(7), 1009–1025 (2011). https://doi.org/10.1016/j.is.2011.04.002. Special Issue: Advanced Information Systems Engineering (CAiSE 2010)