[RSSV25] B. Rumpe, M. Stachon, S. Stüber, V. Voufo:
Tool-Assisted Conformance Checking to Reference Process Models.
DOI 10.48550/arXiv.2508.00738, arXiv, Aug. 2025.

Tool-Assisted Conformance Checking to Reference Process Models

Bernhard Rumpe[1,†], Max Stachon[1,†], Sebastian Stüber[1,†] and Valdes Voufo[1,†]

[1]*Software Engineering, RWTH Aachen University, Germany*

Abstract

Reference models convey best practices and standards. The reference frameworks necessitate conformance checks to ensure adherence to established guidelines and principles, which is crucial for maintaining quality and consistency in various processes. This paper explores automated conformance checks for concrete process models against reference models using causal dependency analysis of tasks and events. Existing notions of conformance checking for process models focus on verifying process execution traces and lack the expressiveness and automation needed for semantic model comparison, leaving this question unresolved. We integrate our approach into a broader semantic framework for defining reference model conformance. We outline an algorithm for reference process model conformance checking, evaluate it through a case study, and discuss its strengths and limitations. Our research provides a tool-assisted solution enhancing accuracy and flexibility in process model conformance verification.

Keywords

Reference Models, Process Models, Semantic, Conformance, BPMN, Model-Driven Engineering

1. Introduction

Process models play a pivotal role across various domains by providing structured representations of workflows and operations. They are instrumental in standardizing processes [1, 2], optimizing performance [3], and ensuring compliance with industry standards [4, 5]. As organizations increasingly rely on these models to guide their operational practices [6], the demand for robust mechanisms to verify the adherence of concrete implementations to reference models has become paramount.

Reference models serve as authoritative benchmarks that encapsulate standardized processes and best practices [7, 8]. In the realm of process modeling, they provide a critical framework against which concrete process implementations can be rigorously evaluated. Despite their significance, there is a notable lack of formal conformance verification methods tailored to facilitate appropriate comparisons between specific process models and more general reference models. Current approaches to conformance checking in process modeling often fall short in addressing the nuanced causal dependencies inherent in both reference and concrete models. Instead, they predominantly focus on verifying execution logs [9, 10, 11, 12], thereby neglecting the essential aspect of model-to-model comparison.

Conformance checks [13] are essential for verifying that concrete process models align with their corresponding reference models, ensuring adherence to best practices and facilitating the timely identification of deviations. This alignment is vital for organizations seeking compliance with regulatory standards, optimizing operational efficiency, and maintaining quality assurance. However, conformance checking presents significant challenges due to the inherent complexity and variability of real-world processes. For instance, the dynamic nature of business operations can lead to frequent changes in process structures, complicating validation efforts. Existing methods of semantic model comparison often struggle with limited expressiveness and scalability, which can significantly hinder effective validation in diverse and complex environments [14, 15, 16, 17, 18].

arxiv.com

[†]These authors contributed equally.

rumpe@se.rwth-aachen.de (B. Rumpe); stachon@se-rwth.de (M. Stachon); stueber@se-rwth.de (S. Stüber); valdes.voufo@rwth-aachen.de (V. Voufo)
https://se-rwth.de (B. Rumpe)
0000-0002-2147-1966 (B. Rumpe); 0000-0002-6328-3816 (M. Stachon); 0000-0002-6636-9375 (S. Stüber)

This paper addresses these limitations by introducing an innovative algorithmic approach for tool-assisted conformance checking that leverages causal relations analysis to enhance the expressiveness and automation of the validation process. Our motivation arises from the pressing need for more accurate and reliable conformance checks, particularly in scenarios involving intricate process structures and dependencies.

The objective of this study is to develop a novel method that systematically analyzes causal dependencies within reference process models and compares them against corresponding elements in concrete models. By doing so, we aim to provide a more flexible and comprehensive solution for conformance verification, thereby advancing the state of the art in process model validation.

Contribution

- Semantic concept for reference process models and conformance
- Abstract description of a conformance checking algorithm
- Publicly available Java implementation for conformance checking
- Evaluation of tool on multiple examples and discussion of results

Structure In Section 2, we present a motivating example that is used to illustrate key concepts. Section 3 provides background information and discusses related work in the field. We then introduce an abstract description of our conformance checking algorithm in Section 4 and address its complexity, soundness, and completeness. Following this, Section 5 details the implementation aspects. The evaluation of our tool is presented in Section 6. In Section 7, we discuss precision and limitations of our approach, as well as potential threads to validity. Finally, we conclude with a summary of our findings and suggest directions for future work.

2. Motivating Example

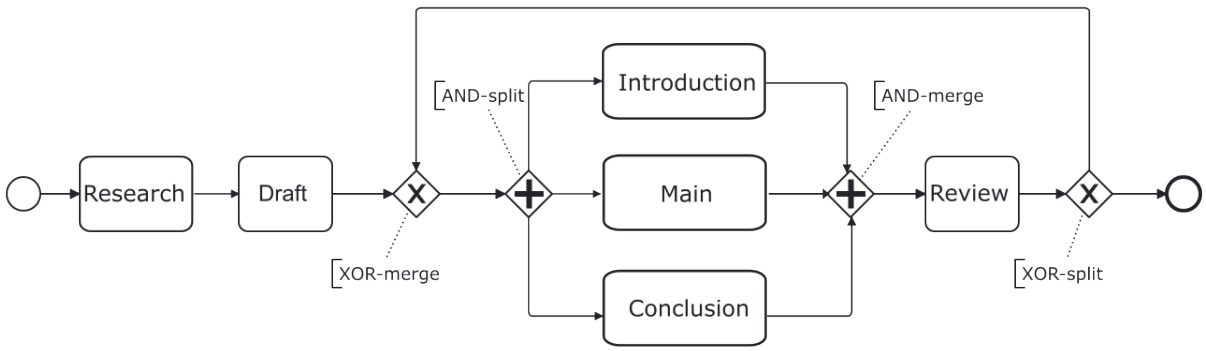

Figure 1: Reference process model for scientific writing.

Consider the process model displayed in fig. 1, it specifies a reference process for scientific writing. After starting the process, the first task is `Research`. After `Research` 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 scientific paper for a conference. This model is displayed in fig. 2: The thesis starts with a `Literature Review` instead of `Research`, followed by writing an `Exposé` instead of

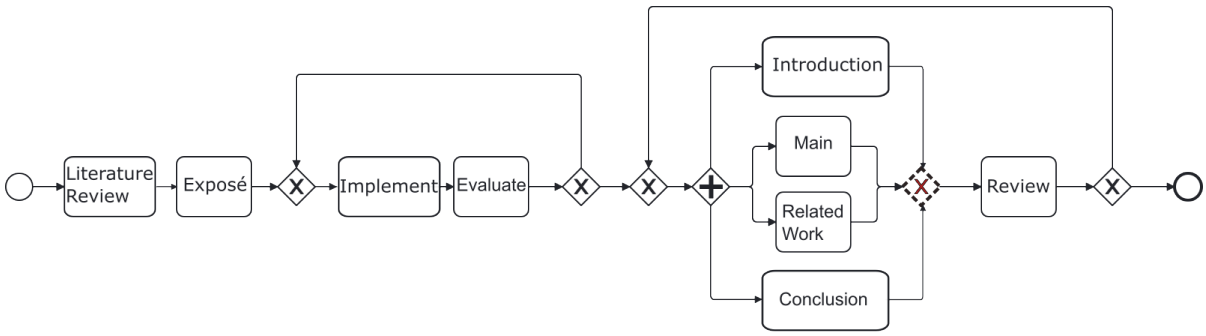

Figure 2: Concrete process model for writing a thesis; the fifth gateway is incorrect and leads to non-conformance.

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, on a closer look at the model displayed in fig. 2 one might discover a mistake made by the modeler: The fifth gateway is an `XOR-merge` instead of the intended `AND-merge`. As a consequence, the execution behavior of the model does not correspond to the reference model, *i.e.,* its semantics was not properly preserved.

A mistake such as this becomes harder to discover the larger the model grows, increasing 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.

This mistake in particular does not cause a deadlock, and can therefore not be discovered by a deadlock-analysis of the model. It might potentially be discovered during testing; after all, merging sequence flows that are executed in parallel might be considered an anti-pattern and corresponding tests might exist. However, tests would have to be implemented for every well-known anti-pattern and even then there are cases were a mistake does not produce an anti-pattern but still alters the behavior of the concrete model in an unintended way, *e.g.,* accidentally placing tasks in a sequence flow in the wrong order. Instead, to detect mistakes such as this that lead to non-conformance with a reference model, we propose the use of a tool-assisted conformance check.

3. Background and Related Work

In the following we outline necessary background information and discuss related work.

Process Modeling with BPMN

Business Process Model and Notation (BPMN) [19] 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 concentrate 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 the purposes

of our study, 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 execution recorded in event logs with the behavior specified by a process model, *i.e.,* the intended or required procedure. This is accomplished by aligning traces from the event log with sequences allowed by the process model, and identifying where the real executions deviate from the model specifications.

While traditional conformance checking focuses on the alignment of actual process executions with a predefined model [9, 10, 11, 12, 20, 21, 22], our investigation shifts towards a theoretical framework that evaluates the structural and semantic alignment between two models: one abstract and one concrete. In this paper, we are concerned with a different kind of conformance relation, which pertains to model-to-model conformance involving a reference model and a concrete model. Here, the focus is not on whether a process execution trace represents a legal instance of a model but rather on whether all legal executions of the concrete model conform to the reference model.

A reference model is defined as an abstract representation within a given modeling language that captures domain concepts and relationships, specifying properties that must hold for any conformant concrete model, *e.g.,* which model elements may or must exist and how they interrelate [23]. A robust conformance relation must guarantee that all relevant semantic properties are preserved by conformant concrete models Konersmann et al. [13]. More 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 and have been developed for various modeling paradigms, including class diagrams, feature models, and statecharts [13]. These algorithms facilitate the evaluation of how well a concrete model adheres to its reference counterpart and determine conformance violations occurring in the concrete model.

The incarnation mappings for these algorithms were either specified using stereotypes [24, 25], 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.

Semantic Differencing

A denotational semantics definition $sem : M \rightarrow 2^D$ assigns each syntactically correct model of a modeling language M to 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, the valid instances correspond to process execution traces.

To enable concrete models to extend a reference model in a meaningful manner, 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 relationships among tasks and events in the reference model are respected in their respective implementations. It is essential to note that we assume tasks and events are uniquely identified by names within a process model, ensuring that each task and event appears only once.

One effective approach to establish a conformance relation is through semantic differencing [27], which analyzes the differences between two models based on their legal instances. Various semantic differencing operators have been developed for multiple modeling languages [15, 28, 29, 30, 16, 31, 32, 33, 34, 17, 18]. The conformance checker for feature diagrams [13] builds upon a prior semantic differencing approach [32] by integrating incarnation mappings.

One approach to formalizing the execution semantics of process models, particularly Business Process Model and Notation (BPMN), is to translate them into Petri nets [35]. However, comparing Petri nets based on their execution traces is generally undecidable [36]. Consequently, existing semantic differencing operators for process models [28, 16] 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, [28] employs this translation to perform bisimulation of the models, while [16] utilizes language inclusion checking algorithms for finite word automata.

However, a significant drawback of the translation to state machines is the exponential growth of the state space compared to the original activity diagram. This power-set automaton construction, which leads to scalability issues, is necessary to capture the semantics of concurrent activities, which must be interleaved in all possible configurations. Alternative, more sophisticated semantic models for concurrency, such as partially ordered multisets (pomsets) and Mazurkiewicz traces [37, 38, 39], exist. These models mitigate the interleaving explosion problem by focusing on partial orders or equivalence classes of sequences that preserve only causal dependencies and independence. Nonetheless, they introduce additional mathematical complexity and necessitate specialized algorithms for determining behavioral equivalences, such as concurrency-aware bisimulation.

Another aspect that complicates the use of bisimulation for conformance checking is the handling of multiple and composite incarnations of reference tasks and events. In this context, multiple incarnations occur when a single element in the reference model is incarnated multiple times in the concrete model, while composite incarnations involve a configuration where a single reference element is represented by multiple elements in the concrete model, or vice-versa. For instance, in process models, multiple incarnations could manifest as inclusive or exclusive alternatives of tasks or events, whereas composite incarnations would necessitate the parallel or sequential execution of these tasks and/or events.

The challenge lies in the requirement that bisimulation establishes a one-to-one correspondence between states in the reference and concrete models. This requirement becomes problematic when dealing with multiple and composite incarnations, as they introduce a level of 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, which complicates the conformance checking process further. 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. Subsequently, we verify whether these dependencies are maintained for the corresponding incarnations in the concrete model.

4. Conformance Checking Approach

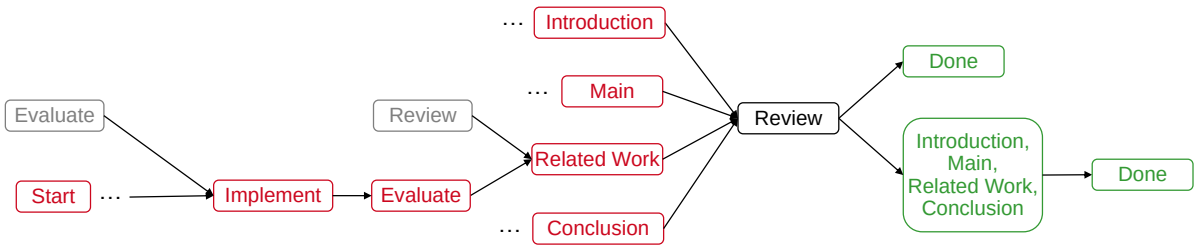

Figure 3: Abbreviated search tree for forward and backward search for the task `Review` in the concrete model in fig. 2 with satisfying branches in green, non-satisfying in red, and branches deleted due to loops / idleness in grey.

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 among 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 correlates each incarnation in the concrete process model to its corresponding task or event in the reference model. Notably, a reference task or event may have multiple incarnations. We analyze each reference task and its incarnations individually, scrutinizing their local causal dependencies with respect to both their 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 methodology for class diagrams as described in [13], 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 instance, 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 in a sequential manner. 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 relationships of each incarnation of a task or event with their predecessors and successors in the concrete model, subsequently comparing these relationships 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 their 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 pseudo-code 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 $AND\big(\{\text{SucForm}(s)\text{: } s \in x.\text{suc}\}\big)$
    else if x is an XOR-split gateway **then**
      return $XOR\big(\{\text{SucForm}(s)\text{: } s \in x.\text{suc}\}\big)$
    else if x is an OR-split gateway **then**
      return $OR\big(\{\text{SucForm}(s)\text{: } s \in x.\text{suc}\}\big)$
    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 pseudo-code for this process can be found in algorithm 2.

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

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

```
Introduction AND Main AND Conclusion
```

If we consider the task `Review`, instead, then we get this same formula but for the backwards 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 respectively its direct predecessors and its direct successors, we perform a quasi-simulation of the concrete model—both forward and backwards—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 pseudo-code 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 pseudo-code for simplicity.

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 in which a branch that was previously marked as *conform* no longer satisfies the formula. As we will elaborate later, this does

not necessarily imply that the execution trace represented by this branch is non-conformant; therefore, we designate its status as *unknown*.

When we encounter an XOR-split gateway in A, we create a new branch for each successor node, updating the sets of visited and active nodes, as well as the corresponding result for each new branch. Subsequently, we delete the current branch. In a similar manner, if an OR-split gateway is encountered in A, we generate a new branch for each subset of successor nodes and then delete the current branch.

Conversely, if the set of active nodes A contains only AND-merge gateways, we can progress through one of these gateways, provided that all its predecessor nodes are included in N. If no such gateway exists, we will advance through another available AND-merge gateway.

Finally, if no active nodes remain but we have not yet reached an end event or returned to n, we delete the branch.

Once all branches with active nodes have been exhausted, we return the set of branches that were not deleted and examine their results. If any non-conformant branch exists, we conclude that the incarnation is *not conform*, and we return 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 as a potential diff witness for manual verification. Lastly, if all remaining branches are conformant, we classify the incarnation as *conform*.

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 conform with regards to its successors. However, this is not the case with regards 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 conform.

The abbreviated search tree for both the forward and backward search is displayed 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 again and the branch will be ignored.

Algorithm 3 Simplified conformance checking algorithm for an incarnation n

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

$\mathcal{B} \leftarrow \{(\emptyset, \{n\}, \text{not conform})\}$
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 conform and N satisfies the formula **then**
    $r =$ conform
  else if $r =$ conform and N does not satisfies the formula **then**
    $r =$ unknown
  end if
end function

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 [28, 16], 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 entails that every execution trace of the concrete model must be permissible under the reference model. Operating under an open-world assumption allows for extensions of the process in the concrete case, which necessitates that the relative order of elements in a trace corresponds to the order of elements in the reference model.

Preserving causal dependencies for incarnations—as implemented by our algorithm—serves as a sufficient condition for semantic refinement under an open-world assumption. At any point during the execution of the concrete process, it is guaranteed that for each active incarnation, all predecessor incarnations can ultimately be identified through backtracking. Furthermore, a suitable configuration of successor incarnations will eventually be established if the execution is continued. Consequently, the resulting execution trace must be permissible according to the reference model.

Completeness: The reduced complexity of our approach, as compared to previous semantic differencing methods for process models [14, 16], does come 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.

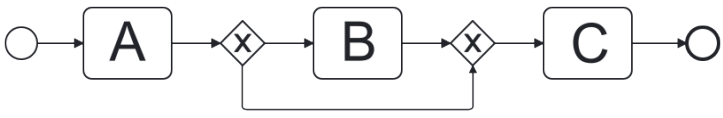

Figure 4: Example of a process model with a skippable task.

5. Implementation & Tool

For our implementation, we make use of a textual version of BPMN [40] developed with the `MontiCore`[1] language workbench [41, 25]. To encode the incarnation mapping, we have extended the grammar of the language to allow the annotation of tasks and events with stereotypes [24, 25]. Similar to our previous approach for conformance checking of class diagrams [13], incarnations are identified via a stereotype that specifies the name of the incarnation mapping as well as the name of the corresponding

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

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 shows a process model in the textual BPMN syntax that describes a sequential writing process. The tasks `Concept` and `Implementation` are each 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 is in fact conform to our reference model for scientific writing displayed in fig. 1, since the order of tasks has simply been sequentialized.

```
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: Example – BPMN in textual syntax

The conformance-checking tool has been integrated in to 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` as demonstrated in listing 2. 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 $-m$.

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

Listing 2: Executing the BPMN conformance checker

Given the example from listing 1 as a concrete model and a textual version of our scientific writing process from fig. 1 as reference model, the conformance check produces the output displayed in listing 3 on the console.

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

--- Final Result of Conformance Checking ---
--- All nodes conform to their reference ---
```

Listing 3: Output in case of conformance

If instead we consider the thesis-writing process from fig. 2 as our concrete model, the tool informs us that the task `Review` is not conform with respect to its predecessors. It also provides a backtracking sequence from `Review` to the start event that demonstrates this, as can be seen in listing 4.

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

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

--- Final Result of Conformance Checking ---
The following nodes do not conform: [Review]

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

Result: Node [AntiPattern:Review] does not conform to Node [PaperAuthoring:Review]
Counter example: The following backtrack [Review, Introduction, Evaluate, Implement, Expose,
    LiteratureReview, Start] is possible in [AntiPattern] but not in [PaperAuthoring].
```

Listing 4: Output in case of non-conformance

Of course there is also the previously discussed case in which the algorithm is unable to determine conformance or non-conformance. This happens, *e.g.*,, if we check conformance of the process model from fig. 4 to itself. In this particular situation, the tool determines that the task A may not be conform with regards to its successors. It then produces a potentially non-conformant sequence from A to the end event, as can be seen in listing 5.

```
Checking Conformance of [Concrete:Skip] to [Reference:Skip]

--- Final Result of Conformance Checking ---
The status of the following nodes is unknown: [A]

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

Result: Node [Skip:A] may not conform to Node [Skip:A]
Counter example: The following run [B, C, Done] is possible in [Skip] but may not be possible in [
    Skip].
```

Listing 5: Output in case of potential non-conformance

6. Evaluation

In order to evaluate the implementation of our algorithm, 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 *conform*, and ten that are non-conform. We execute the conformance check to compare the modified model to the original and verify the results. The tests for the case study can be found in the BPMN project at:

`WorkflowConformance/src/test/java/`
`de/monticore/bpmn/conformance/CaseStudyTest.java`

with the corresponding model-files at:

`WorkflowConformance/src/test/resources/`
`de/monticore/bpmn/conformance/caseStudy/`

6.1. Conformant Modifications

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

In all cases the tool informs us that all nodes are conform.

6.2. Non-Conformant Modifications

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

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.

7. Discussion

In this study, we explored the concept of conformance checking of concrete models to reference models within process modeling languages such as BPMN. We developed an approach that focuses on preserving causal dependencies among tasks and events in the context of their incarnations. In this section, we will discuss aspects of uncertainty whithin our approach, its limitations, and potential threats to validity.

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

In cases where the algorithm determines the non-conformance of an incarnation, it uncovers a path within the model that violates the local dependency of tasks and events as specified by the reference model. AProvided that the incarnation is reachable and a path to an end event exists, this path can be expanded into a process execution trace that acts as a *diff witness*, *i.e.,* a trace that is not permitted by the reference model, while operating under the open-world assumption and in the context of incarnation.

Uncertainty therefore only exists in the form of incarnations whose conformance status cannot be determined and are hence labeled as *unknown*. This occurs when a satisfying configuration is followed by a non-satisfying configuration in a subsequent step in the same branch of the forward search, which in turn can only result from visiting incarnations of exclusive alternatives among successors.

Unlike previous conformance checking approaches that exhibit uncertainty regarding semantic refinement [13], our method cannot simply categorize these instances as non-conformant. This is because reflexivity is a necessary component of a conformance relation, and as illustrated in fig. 4, labeling them as non-conformant would not be justified. Consequently, we recognize our approach as somewhat incomplete, necessitating a manual review of these ambiguous cases. Fortunately, our algorithm provides both the name of the individual incarnation and the execution trace that is flagged as potentially non-conformant, facilitating this review process.

Limitations: A significant limitation of our current approach is that it only considers a subset of BPMN language features [19]. Specifically, we focus on basic tasks, events, and the logical gateways XOR, OR, and AND, which together represent a rudimentary foundation for process modeling. Furthermore, events are treated similarly to tasks, with our examples restricted to start and end events. Future work should explore additional BPMN features and their implications for process semantics.

Certain features, such as lanes, are not integrated into our current trace-based semantics definition and may therefore be overlooked. However, elements like expressions—especially executable formal expressions—play a crucial role in defining flow and loop conditions, as well as event triggers. Our textual representation of BPMN accommodates such formal expressions, and future enhancements to the algorithm could involve conformance checking of these expressions.

One promising approach would be to utilize SMT-solving to verify the refinement of individual reference expressions against their concrete counterparts. Additionally, sub-processes could potentially be managed through a hierarchical conformance check, while call activities may be addressed similarly.

Moreover, data and message flows represent another critical aspect for consideration. In our BPMN implementation, usable data types can be defined using class diagrams [42, 43], allowing us to leverage our existing conformance checks [13]. If behaviors are specified through operation constraints in OCL [44, 18], operations declared in a class diagram and referenced in a task of the process model could also be subjected to conformance verification.

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 the necessary expressiveness. For instance, it does not allow for a task in the reference model to be represented as a sequence of tasks. To address these limitations, we propose the development of a custom mapping language that can enhance the expressiveness and versatility of our incarnation mappings.

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

For instance, we explicitly disallow the addition of loops that involve tasks and events present in the model by necessitating suitable configurations of direct successor and predecessor tasks and events between two instances of the same task or event within an execution trace. In some scenarios, however, these loops may represent relevant refinement steps. It might be sufficient for the aforementioned predecessor and successor configurations to occur just once—respectively, before and after all instances of the task or event within the trace.

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

8. Conclusion

This paper presents an innovative approach to conformance checking of reference process models through the introduction of an algorithmic solution that leverages causal relations analysis. Our primary motivation was to enhance both the expressiveness and automation of conformance checks, enabling more precise verification of complex process structures. We achieved this objective by developing a novel method that systematically examines causal interdependency of elements within reference models and compares it with the interdependency of corresponding elements in concrete models, thereby providing a more adaptable and comprehensive framework for conformance verification.

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

Looking ahead, future work will involve conducting industry case studies to validate our approach in real-world scenarios, as well as extending our BPMN feature support to encompass a broader range of elements and language variants [45, 46]. These efforts aim to further enhance the practical applicability and robustness of our method, ensuring its relevance across diverse industrial contexts.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 250902306

References

[1] M. C. Ungan, Standardization through process documentation, Business Process Management Journal 12 (2006) 135–148. doi:10.1108/14637150610657495.

[2] T. Allweyer, Bpmn 2.0: introduction to the standard for business process modeling, BoD–Books on Demand, 2016.

[3] G. Kougka, A. Gounaris, A. Simitsis, The many faces of data-centric workflow optimization: a survey, International Journal of Data Science and Analytics 6 (2018) 81–107. doi:10.1007/s41060-018-0107-0.

[4] D. Knuplesch, M. Reichert, R. Pryss, W. Fdhila, S. Rinderle-Ma, Ensuring compliance of distributed and collaborative workflows, in: 9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, IEEE, 2013, pp. 133–142.

[5] H. W. Lim, F. Kerschbaum, H. Wang, Workflow signatures for business process compliance, IEEE Transactions on Dependable and Secure Computing 9 (2012) 756–769.

[6] J. Fernandes, J. Reis, N. Melão, L. Teixeira, M. Amorim, The role of industry 4.0 and bpmn in the arise of condition-based and predictive maintenance: A case study in the automotive industry, Applied Sciences 11 (2021) 3438.

[7] ITU-T, Information technology – Open Systems Interconnection – Basic Reference Model: The basic model, ITU-T X.200, International Tecommunication Union, 1994.

[8] E. Gamma, R. Helm, R. E. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Prentice Hall, 1997.

[9] A. Rozinat, W. M. P. van der Aalst, Conformance testing: Measuring the fit and appropriateness of event logs and process models, in: C. J. Bussler, A. Haller (Eds.), Business Process Management Workshops, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 163–176.

[10] W. Van der Aalst, A. Adriansyah, B. Van Dongen, Replaying history on process models for conformance checking and performance analysis, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2 (2012) 182–192.

[11] A. Burattin, F. M. Maggi, A. Sperduti, Conformance checking based on multi-perspective declarative process models, Expert Systems with Applications 65 (2016) 194–211. URL: https://www.sciencedirect.com/science/article/pii/S0957417416304390. doi:https://doi.org/10.1016/j.eswa.2016.08.040.

[12] S. Dunzer, M. Stierle, M. Matzner, S. Baier, Conformance checking: a state-of-the-art literature review, in: Proceedings of the 11th International Conference on Subject-Oriented Business Process Management, S-BPM ONE '19, Association for Computing Machinery, New York, NY, USA, 2019. URL: https://doi.org/10.1145/3329007.3329014. doi:10.1145/3329007.3329014.

[13] M. Konersmann, B. Rumpe, M. Stachon, S. Stüber, V. Voufo, Towards a Semantically Useful Definition of Conformance with a Reference Model, Journal of Object Technology (JOT) 23 (2024) 1–14. URL: http://www.se-rwth.de/publications/Towards-a-Semantically-Useful-Definition-of-Conformance-with-a-Reference-Model.pdf. doi:10.5381/jot.2024.23.3.a5.

[14] S. Maoz, J. O. Ringert, B. Rumpe, An Operational Semantics for Activity Diagrams using SMV, Technical Report AIB-2011-07, RWTH Aachen University, Aachen, Germany, 2011. URL: http://www.se-rwth.de/publications/An-Operational-Semantics-for-Activity-Diagrams-using-SMV.pdf.

[15] S. Maoz, J. O. Ringert, B. Rumpe, CDDiff: Semantic Differencing for Class Diagrams, in: M. Mezini (Ed.), ECOOP 2011 - Object-Oriented Programming, Springer Berlin Heidelberg, 2011, pp. 230–254. URL: https://se-rwth.de/publications/CDDiff-Semantic-Differencing-for-Class-Diagrams.pdf.

[16] O. Kautz, B. Rumpe, Semantic Differencing of Activity Diagrams by a Translation into Finite Automata, in: Proceedings of MODELS 2018. Workshop ME, 2018. URL: http://www.se-rwth.de/publications/Semantic-Differencing-of-Activity-Diagrams-by-a-Translation-into-Finite-Automata.pdf.

[17] J. O. Ringert, B. Rumpe, M. Stachon, On Implementing Open World Semantic Differencing for Class Diagrams, Journal of Object Technology (JOT) 22 (2023) 2:1–14. URL: http://www.se-rwth.de/publications/On-Implementing-Open-World-Semantic-Differencing-for-Class-Diagrams.pdf. doi:10.5381/jot.2023.22.2.a11.

[18] B. Rumpe, M. Stachon, S. Stüber, V. Voufo, Semantic Difference Analysis with Invariant Tracing for Class Diagrams Extended by OCL, in: Workshop on Model Driven Engineering, Verification and Validation, MODELS Companion '24: International Conference on Model Driven Engineering Languages and Systems (MoDeVVa), Association for Computing Machinery (ACM), 2024, p. 1066â€“1075. URL: http://www.se-rwth.de/publications/Semantic-Difference-Analysis-with-Invariant-Tracing-for-Class-Diagrams-Extended-by-OCL.pdf. doi:10.1145/3652620.3687818.

[19] OMG, Business process model and notation, 2014. URL: https://www.omg.org/spec/BPMN/2.0.2/PDF.

[20] M. Rafiei, W. M. van der Aalst, Mining roles from event logs while preserving privacy, in: International Conference on Business Process Management, Springer, 2019, pp. 676–689.

[21] D. Schuster, G. J. Kolhof, Scalable online conformance checking using incremental prefix-alignment computation, in: International Conference on Service-Oriented Computing, Springer, 2020, pp. 379–394.

[22] M. Rafiei, M. Pourbafrani, W. M. van der Aalst, Federated conformance checking, Information Systems 131 (2025) 102525.

[23] M. Konersmann, J. Michael, B. Rumpe, Towards Reference Models with Conformance Relations for Structure, Logos Verlag Berlin, 2024, pp. 247–269. URL: http://www.se-rwth.de/publications/Towards-Reference-Models-with-Conformance-Relations-for-Structure.pdf.

[24] M. Gogolla, B. Henderson-Sellers, Analysis of UML Stereotypes within the UML Metamodel, Springer Berlin Heidelberg, 2002, pp. 84–99. doi:10.1007/3-540-45800-x_8.

[25] K. Hölldobler, O. Kautz, B. Rumpe, MontiCore Language Workbench and Library Handbook: Edition 2021, Aachener Informatik-Berichte, Software Engineering, Band 48, Shaker Verlag, 2021. URL: http://www.monticore.de/handbook.pdf.

[26] D. Harel, B. Rumpe, Meaningful Modeling: What's the Semantics of "Semantics"?, IEEE Com-

puter Journal 37 (2004) 64–72. URL: http://www.se-rwth.de/staff/rumpe/publications20042008/Meaningful-Modeling-Whats-the-Semantics-of-Semantics.pdf.

[27] S. Maoz, J. O. Ringert, B. Rumpe, A Manifesto for Semantic Model Differencing, in: Proceedings Int. Workshop on Models and Evolution (ME'10), LNCS 6627, Springer, 2010, pp. 194–203. URL: http://www.se-rwth.de/publications/A-Manifesto-for-Semantic-Model-Differencing.pdf.

[28] S. Maoz, J. O. Ringert, B. Rumpe, ADDiff: Semantic Differencing for Activity Diagrams, in: Conference on Foundations of Software Engineering (ESEC/FSE '11), ACM, 2011, pp. 179–189. URL: http://www.se-rwth.de/publications/ADDiff-Semantic-Differencing-for-Activity-Diagrams.pdf.

[29] P. Langer, T. Mayerhofer, G. Kappel, Semantic model differencing utilizing behavioral semantics specifications, in: J. Dingel, W. Schulte, I. Ramos, S. Abrahão, E. Insfran (Eds.), Model-Driven Engineering Languages and Systems, Springer International Publishing, Cham, 2014, pp. 116–132.

[30] A. Butting, O. Kautz, B. Rumpe, A. Wortmann, Semantic Differencing for Message-Driven Component & Connector Architectures, in: International Conference on Software Architecture (ICSA'17), IEEE, 2017, pp. 145–154. URL: http://www.se-rwth.de/publications/Semantic-Differencing-for-Message-Driven-Component-and-Connector-Architectures.pdf.

[31] A. Butting, O. Kautz, B. Rumpe, A. Wortmann, Continuously Analyzing Finite, Message-Driven, Time-Synchronous Component & Connector Systems During Architecture Evolution, Journal of Systems and Software (JSS) 149 (2019) 437–461. URL: http://www.se-rwth.de/publications/Continuously-Analyzing-Finite-Message-Driven-Time-Synchronous-Component-and-Connector-Systems-During pdf. doi:https://doi.org/10.1016/j.jss.2018.12.016.

[32] I. Drave, R. Eikermann, O. Kautz, B. Rumpe, Semantic Differencing of Statecharts for Object-oriented Systems, in: S. Hammoudi, L. F. Pires, B. Selić (Eds.), Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD'19), SciTePress, 2019, pp. 274–282. URL: http://www.se-rwth.de/publications/Semantic-Differencing-of-Statecharts-for-Object-oriented-Systems.pdf.

[33] I. Drave, O. Kautz, J. Michael, B. Rumpe, Semantic Evolution Analysis of Feature Models, in: T. Berger, P. Collet, L. Duchien, T. Fogdal, P. Heymans, T. Kehrer, J. Martinez, R. Mazo, L. Montalvillo, C. Salinesi, X. Tërnava, T. Thüm, T. Ziadi (Eds.), International Systems and Software Product Line Conference (SPLC'19), ACM, 2019, pp. 245–255. URL: http://www.se-rwth.de/publications/Semantic-Evolution-Analysis-of-Feature-Models.pdf.

[34] O. Kautz, Model Analyses Based on Semantic Differencing and Automatic Model Repair, Aachener Informatik-Berichte, Software Engineering, Band 46, Shaker Verlag, 2021. URL: http://www.se-rwth.de/phdtheses/Diss-Kautz-Model-Analyses-Based-on-Semantic-Differencing-and-Automatic-Model-Repair.pdf.

[35] R. M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in bpmn, Information and Software technology 50 (2008) 1281–1294.

[36] Y. Hirshfeld, Petri nets and the equivalence problem, in: International Workshop on Computer Science Logic, Springer, 1993, pp. 165–174.

[37] V. Pratt, Modeling concurrency with partial orders, International journal of parallel programming 15 (1986) 33–71.

[38] A. Mazurkiewicz, Trace theory, in: Advanced course on Petri nets, Springer, 1986, pp. 278–324.

[39] B. Bloom, M. Kwiatkowska, Trade-offs in true concurrency: Pomsets and mazurkiewicz traces, in: International Conference on Mathematical Foundations of Programming Semantics, Springer, 1991, pp. 350–375.

[40] I. Drave, J. Michael, E. Müller, B. Rumpe, S. Varga, Model-Driven Engineering of Process-Aware Information Systems, Springer Nature Computer Science Journal 3 (2022). URL: http://www.se-rwth.de/publications/Model-Driven-Engineering-of-Process-Aware-Information-Systems.pdf.

[41] H. Grönniger, H. Krahn, B. Rumpe, M. Schindler, S. Völkel, MontiCore 1.0: Ein Framework zur Erstellung und Verarbeitung domänspezifischer Sprachen, Informatik-Bericht 2006-04, CFG-Fakultät, TU Braunschweig, 2006. URL: http://www.se-rwth.de/staff/rumpe/publications20042008/MontiCore-1.0-Ein-Framework-zur-Erstellung-und-Verarbeitung-domaenenspezifischer-Sprachen.pdf.

[42] A. Haber, M. Look, P. Mir Seyed Nazari, A. Navarro Perez, B. Rumpe, S. Völkel, A. Wortmann, Integration of Heterogeneous Modeling Languages via Extensible and Composable Language Components, in: Model-Driven Engineering and Software Development Conference (MODELSWARD'15), SciTePress, 2015, pp. 19–31. URL: http://www.se-rwth.de/publications/Integration-of-Heterogeneous-Modeling-Languages-via-Extensible-and-Composable-Language-Components.pdf.

[43] A. Haber, M. Look, P. Mir Seyed Nazari, A. Navarro Perez, B. Rumpe, S. Völkel, A. Wortmann, Composition of Heterogeneous Modeling Languages, in: Model-Driven Engineering and Software Development, volume 580 of *Communications in Computer and Information Science*, Springer, 2015, pp. 45–66. URL: http://www.se-rwth.de/publications/Composition-of-Heterogeneous-Modeling-Languages.pdf.

[44] S. Cook, A. Kleppe, R. Mitchell, B. Rumpe, J. Warmer, A. Wills, The Amsterdam Manifesto on OCL, in: T. Clark, J. Warmer (Eds.), Object Modeling with the OCL, LNCS 2263, Springer Verlag, 2002, pp. 115–149. URL: https://www.se-rwth.de/staff/rumpe/publications/The-Amsterdam-Manifesto-on-OCL.pdf.

[45] M. V. Cengarle, H. Grönniger, B. Rumpe, Variability within Modeling Language Definitions, in: Conference on Model Driven Engineering Languages and Systems (MODELS'09), LNCS 5795, Springer, 2009, pp. 670–684. URL: http://www.se-rwth.de/publications/Variability-within-Modeling-Language-Definitions.pdf.

[46] H. Grönniger, B. Rumpe, Modeling Language Variability, in: Workshop on Modeling, Development and Verification of Adaptive Systems. (16th Monterey Workshop), Redmond, Microsoft Research, 2010. URL: https://www.se-rwth.de/publications/Modeling-Language-Variability.Redmond.pdf.