Soundness Verification for Conceptual Workflow Nets with Data: Early Detection of Errors with the Most Precision Possible

Natalia Sidorova*+, Christian Stahl¹, Nikola Trˇcka²

*Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract
A conceptual workflow model specifies the control flow of a workflow together with abstract data information. This model is later on refined by adding specific data information, resulting in an executable workflow which is then run on an information system. It is desirable that correctness properties of the conceptual workflow are transferable to its refinements. In this paper, we present classical workflow nets extended with data operations as a conceptual workflow model. For these nets, we develop a novel technique to verify soundness. An executable workflow is sound if from every reachable state it is always possible to terminate properly. Our technique allows us to analyze a conceptual workflow and to conclude whether there exists at least one sound refinement of it, and whether any refinement of a conceptual workflow model is sound. The positive answer to the first question in combination with the negative answer to the second question means that sound and unsound refinements for the conceptual workflow in question are possible.

Keywords: conceptual workflow models, soundness, refinement, verification, correctness, may/must semantics

1. Introduction
Information systems are a key technology in today’s organizations. Prominent examples of information systems are enterprise resource planning (ERP) systems and workflow management systems. Processes are the core of most information systems [1]. They orchestrate people, information, and technology to deliver products. In this paper, we focus on workflows. A workflow refers to the automation of a processes by an IT infrastructure, in whole or in part [2].

A workflow is usually iteratively designed following a bottom-up approach. First, the control flow of the workflow is modeled. The control flow consists of a set of coordinated tasks describing the behavior of the workflow. Later, the control flow is extended with abstract data information. The resulting model is an abstract or conceptual workflow model; it is typically constructed by a business analyst. This conceptual model can be used for purposes of documentation, communication, and analysis. It may abstract from concrete data values, such as the condition of an if-then-else construct, and it usually does not specify how concrete data values are stored.

To actually execute this workflow on a workflow management system, the conceptual workflow model is instantiated with full detail—a task typically performed by business programmers (who often have insufficient background knowledge of the process) and not by the business analysts themselves. For example, the business programmer specifies concrete data values and how they are stored. Modeling abstract and executable workflows is supported by industrial workflow modeling languages available on the market.

Designing a workflow model is a complicated and error-prone task even for experienced process designers. For a fast and, therefore, cost-efficient design process, it is important that errors in the model are detected during the design phase rather than at runtime. Hence, verification needs to be applied at an early stage—that is, already on the level of the conceptual workflow model.

Formal verification of a conceptual workflow model imposes two challenges. First, it requires an adequate formal model. With adequate we mean that the model captures the appropriate level of abstraction and enables efficient analysis. So the question is, how can we formalize commonly used conceptual workflow models? Second, verification must not be restricted to the control flow, but should also incorporate available data information. Thereby the conceptual workflow model may specify abstract data values that will be refined later on. Here the question is, can we verify conceptual workflows in such a way that the results hold for any possible data refinement (i.e., for all possible executable versions)?

In this paper, we investigate these two questions. We focus on one of the most established requirement for workflow correctness, the soundness property [3]. Soundness guarantees that the workflow has always the possibility to terminate and every task of the workflow is coverable (i.e., can potentially be executed). Termination ensures that the workflow can during its execution neither get stuck (i.e., it is deadlock free) nor enter a loop that cannot be left (i.e., the workflow is livelock free).
whereas coverable excludes dead tasks in the workflow. However, current techniques for verifying soundness are mostly restricted to the control flow and do not consider data information.

To answer the first question, we propose workflow nets with data (WFD-nets) as an adequate formalism for modeling conceptual workflows. A WFD-net is a workflow net (i.e., a Petri net tailored toward the modeling of the control flow of workflows) extended with conceptual read/write/delete data operations and guards. A guard defines an additional constraint that must be fulfilled to enable a transition (i.e., a task). WFD-nets generalize our previous model from [4] by supporting arbitrary guards—previously, only predicate-negation and conjunctions were allowed. The syntax of WFD-nets is actually similar to the syntax of the Protos tool of Pallas Athenas, which is the leading process modeling tool in the Netherlands.

Example 1. We illustrate the idea of modeling conceptual workflows with WFD-nets with the WFD-net in Figure 1, modeling a shipping company. Ignoring the transition guards (shown within squared brackets above transitions), and the read and write operations (denoted by \( r d \) and \( w t \) inside the transitions), Figure 1 shows an ordinary workflow net that consists of ten places (represented as circles) and nine transitions (represented as squares). There are two distinguished places: \( s t a r t \) and \( e n d \). Place \( s t a r t \) models the initial state of the workflow and \( e n d \) the final state.

Initially there is a token in place \( s t a r t \). The shipper receives goods from a client (transition \( r e c e i v e \) goods) to be shipped. In the model, transition \( r e c e i v e \) good writes the data of the client (\( c l \)), the goods (\( g d s \)), and the destination of the goods (\( a d s \)). Then the shipper calculates the price (transition \( c a l c u l a t e \) price). Afterward, the shipment department and the customer adviser execute their tasks concurrently. If the price of the goods is high (i.e., the transition guard \( i s H i g h ( p r i c e ) \) evaluates to true—the exact bound for a price to be high is left unspecified), express shipment is used (transition \( s h i p \) express); otherwise, the cheaper standard shipment is used (transition \( s h i p \) normal). Based on the same condition, the customer adviser either calculates a bonus (transition \( c a l c u l a t e \) bonus) for the client and registers this bonus (transition \( r e g i s t e r \) bonus) or no bonus is calculated (transition \( n o \) bonus). In addition, clients sending goods of a high price are notified about their bonus and the shipment (transition \( i n f o r m \) by \( c a l l \)); other clients receive only a notification of the shipment (transition \( i n f o r m \) by mail).

As a second contribution, we develop a novel technique for analyzing soundness of WFD-nets. The actual challenge is to analyze a WFD-net (i.e., a conceptual workflow) and to conclude from this analysis result whether every data refinement of the WFD-net (i.e., every instantiation of the conceptual workflow with full details) is sound.

First, we study the termination criterion of the soundness property in isolation. Unlike the existing approaches, for example, [4, 5], which could give false positives (i.e., the analysis gives the verdict “sound” although the workflow is actually unsound when the data information is refined) or false negatives (i.e., the analysis gives the verdict “unsound” although the workflow with refined data is sound), our technique gives neither false positives nor false negatives.

Our technique is based on the idea of may-must semantics, as introduced by Larsen and Thomsen [6], that guarantees the analysis results to be valid in any concrete data refinement. If a WFD-net is proven must-sound, then it is sound in any possible data refinement; if it is not may-sound, then no data refinement can make it sound. We define the semantics of WFD-nets using the concept of hyper transition systems [7, 8] rather than standard may/must transition systems [6]. A hyper transition system allows to connect a single state to a set of successor states (referred to as must-set), thereby increasing the precision of the model. The use of hyper transition systems enables us to improve our previous characterizations of may- and must-soundness from [4]. Moreover, we define the must-sets to be minimal ones to achieve the most precise possible, thus improving our result from [9]: We show that if a WFD-net is may-sound, there is at least one sound refinement of it, and if a WFD-net is not must-sound, there is at least one unsound refinement.

Example 2. We illustrate may- and must-soundness using the example in Figure 1. This WFD-net is must-sound; that is, any possible data refinement is sound: Starting with a (colored) token in place \( i n i t i a l \), it is always possible to reach a marking consisting of one token in place \( e n d \). In contrast, if we abstract from data and consider only the underlying workflow net, the net may deadlock. For example, without data the shipment department may decide to use express shipment, but the customer advisor does not calculate any bonus. This yields one token in place \( p 5 \) and one token in place \( p 8 \), and the net gets stuck. This shows that ignoring data information in verification can lead to obtaining false negatives.

Suppose now that instead of the same predicate \( i s H i g h ( p r i c e ) \) two possibly different predicates, say \( i s H i g h L e f t ( p r i c e ) \) and \( i s H i g h R i g h t ( p r i c e ) \), are used in the transition guards of the left and the right parts of the workflow (this is realistic, because these parts correspond to two different departments of the shipper). As we did not change the control flow, the underlying workflow net is still unsound. Our previous work [4] on soundness of (less general) WFD-nets would give the same verdict: The workflow is unsound, because it is possible that the predicates have different truth values. The methods we introduce in this paper enable us to give the correct answer: “I do not know—the workflow is sometimes sound (when \( i s H i g h L e f t ( p r i c e ) \) and \( i s H i g h R i g h t ( p r i c e ) \) always evaluate to the same values), and sometimes it is not (when the valuations of \( i s H i g h L e f t ( p r i c e ) \) and \( i s H i g h R i g h t ( p r i c e ) \) can differ).” In other words, the workflow is may-sound but not must-sound.

It is possible to construct examples in which the verification of workflow nets (without data information) would give a false positive and examples in which our previous work on WFD-nets [4, 5] would give a false negative. For instance, consider the loop in Figure 2 that is exited when some predicate depending on a data element \( d \) evaluates to true. Data element \( d \) is initialized inside this loop (transition \( t 2 \)); it is deleted (denoted
Figure 1: WFD-net modeling a shipper: The workflow is sound, but verification ignoring data leads to the verdict “unsound.”

by del inside transition \( t_4 \) and written in every loop cycle. In this example, proper termination is only possible if \( d \) eventually gets a value on which the exiting predicate \( \text{pred}(d) \) is evaluated to true. Therefore, the correct answer is again, “I do not know,” because the soundness property depends on the concrete data refinement. In other words, the conceptual workflow is may-sound but not must-sound. However, when using workflow nets, the data information is ignored and the possibility to exit the loop is considered to be available, which results in the verdict that the workflow net is sound. Similarly, the previous WFD-net method in [4, 5] would discover that there is a possible faulty execution sequence (namely the infinite looping) and would report unsoundness.

The definition of soundness in [3] requires, in addition to termination, every transition to be coverable (i.e., the workflow has no dead tasks). Although we want to incorporate this coverability requirement, we cannot simply add it in the context of conceptual workflow nets. Intuitively, some refinements might consciously forbid some of the behavioral branches of the conceptual workflow net reflecting, for example, the current market situation. Later on, this choice might be changed. Thus, even a must-sound WFD-net may contain transitions which are not coverable in all data refinements, still remaining a correct, intended design. At the same time, for some transitions it can be essential to be coverable in all data refinements; for example, for some processes it is necessary that the payment transition is potentially executable in every possible data refinement (which does not imply that the payment transition does fire in every run of every data refinement). Therefore, we relax the coverability requirement of [3] by requiring selected rather than all transitions in conceptual workflow nets to be coverable in at least one or in every data refinement. We provide definitions and correctness criteria for these two notions of coverability, which enable precise diagnostics on the level of WFD-nets.
Example 3. In the WFD-net in Figure 1, transitions receive goods and calculate price are coverable in any data refinement; all other transitions can only be executed in some data refinements, because their execution depends on the concrete evaluation of the predicate isHigh(price).

To sum up, the main contributions of this paper are:

- A generalized model for conceptual workflows compared to our previous work in [4];
- An improvement of the analysis result for soundness (without coverability) of [9], thereby achieving the most precise possible; and
- A decision procedure to analyze soundness extended with a coverability requirement while achieving the most precise analysis result possible.

Organization of the paper. We continue by introducing the conceptual workflow model, WFD-nets, in Section 2 and its semantics in Section 3. In Section 4, we define executable workflow nets with data (eWFD-nets) to model refinements of WFD-nets and show the links between the behavior of WFD-nets and their refinements. In Section 5, we define soundness (restricted to termination) for WFD-nets in the may approach. In Section 6, Section 7 presents the related works. Finally, we conclude and discuss future work in Section 8.

2. Workflow nets with data

Given the motivation for incorporating data into workflow nets, this section formalizes workflow nets with data.

2.1. Petri nets

WFD-nets extend workflow nets, a class of Petri nets tailored toward the modeling of workflows; so we define Petri nets first.

Definition 1 (Petri net). A Petri net $N = (P, T, F)$ consists of two disjoint nonempty finite sets $P$ of places and $T$ of transitions and of a flow relation $F \subseteq (P \times T) \cup (T \times P)$.

For a transition $t \in T$, we define the preset of $t$ as $\downarrow t = \{p | (p, t) \in F\}$ and the postset of $t$ as $\uparrow t = \{p | (t, p) \in F\}$. Analogously, we define for a place $p \in P$ its preset $\downarrow p = \{t | (p, t) \in F\}$ and its postset $\uparrow p = \{t | (t, p) \in F\}$. A place $p$ is a source place if $\downarrow p = \emptyset$ and a sink place if $\uparrow p = \emptyset$. We interpret presets and postsets as multisets when used in operations with multisets.

At any time, a place may contain tokens. A token is represented as a black dot. A state of a Petri net, called a marking, is a distribution of tokens over its places. Formally, a marking is defined as a mapping $m : P \rightarrow \mathbb{N}$—that is, as a multiset over $P$. We use standard notation for multisets and write, for example, $m = [2p, q]$ for a marking $m$ with $m(p) = 2$, $m(q) = 1$, and $m(x) = 0$ for $x \in P \setminus \{p, q\}$. We define $+$ and $-$ for the sum and the difference of two markings as $=, <, >, \leq, \geq$ for comparison of markings in the standard way. For the previously defined marking $m$ we have, for instance, $m = [3p, 2q, r]$ and $m + [q, 3r] = [2p, 2q, 3r]$. A pair $(N, m)$, where $N$ is a Petri net and $m$ is a marking of $N$, is a marked Petri net.

The behavior of a Petri net $N$ relies on the marking of $N$ and changing the marking by the firing of transitions of $N$. A transition $t \in T$ is enabled at a marking $m$ of $N$, denoted by $m \xrightarrow{t}$, if $m \geq t$. An enabled transition $t$ may fire, which results in a new marking $m'$ defined by $m' = m - t + t'$. This firing is denoted as $m \xrightarrow{t} m'$. A marking $m'$ is reachable from a marking $m$ if there exists a sequence of firings $m_0 \xrightarrow{t_1} \ldots \xrightarrow{t_n} m_n$ such that for $1 \leq i < n, t_i \in T, m_1 = m, m' = m_n$. We refer to such a sequence of firings as a run of $N$. If a reachable marking $m$ enables a transition $t \in T$, then $t$ is coverable in $N$ (i.e., $t$ is fireable in at least one run of $N$).

2.2. Workflow nets

Workflows are case-based; that is, every piece of work is executed for a specific case. One can think of a case as a workflow instance, such as a mortgage, an insurance claim, or a purchase order. Each case is handled individually according to the workflow definition. The workflow definition specifies which tasks need to be executed for a case and in what order. The order, in which tasks are executed, is determined by conditions specifying dependencies between tasks.

We can model a workflow definition as a Petri net, thereby modeling tasks by transitions, conditions by places, and cases as tokens. The assumption that a typical workflow has a well-defined starting point and a well-defined ending point imposes syntactic restrictions on Petri nets that resulted in the following definition of a workflow net [3].

Definition 2 (WF-net). A Petri net $N = (P, T, F)$ is a workflow net (WF-net) if it has a single source place start, a single sink place end, and every place and every transition is on a path from start to end (i.e., $(\text{start}, n) \in F^*$ and $(n, \text{end}) \in F^*$, for all $n \in P \cup T$, where $F^*$ is the reflexive-transitive closure of $F$).

For an overview of workflows and workflow modeling with Petri nets, we refer to Van der Aalst and Van Hee [2].
A case in its initial state refers to an instance of the workflow definition and, therefore, is modeled by a marked WF-net in which place start has one token and all other places are empty. Executing a workflow means creating a running instance of this workflow (i.e., a case). Several cases of a workflow may coexist. Cases are assumed to be independent of each other—they only possibly share resources. Hence, each case is modeled as a copy of the corresponding workflow net \( N \) with a token in place start. We refer to [start] as the initial marking of N and to the properties of \( \langle N, [\text{start}] \rangle \) as the properties of \( N \).

**Example 4.** Ignoring the transition guards and the read and write operations, Figure 1 depicts a WF-net. Place start is the source place, and place end is the sink place. Clearly, every place and transition is on a path from start to end. The preset of transition Inform by call is the set \( \{p5,p7\} \), and the postset of this transition is the set \( \{\text{end}\} \).

Each transition, such as Inform by call, models a task and places model the dependencies between tasks. Adding a token to place start models a case of the shipper process in its initial state. Every reachable marking corresponds to a state of this respective case.

One of the most established correctness properties of WF-nets is soundness, as introduced by Van der Aalst [3]. Verification of soundness, as introduced by Van der Aalst et al. [10] for an overview. The soundness property comprises three requirements:

1. **Option to terminate:** For every marking \( m \) reachable from the initial marking [start], there is a firing sequence leading from \( m \) to the final marking [end];

2. **Proper termination:** The only reachable marking that marks the sink place is the marking [end]; and

3. **Absence of dead transitions:** For every transition \( t \), there is a marking \( m \) reachable from the initial marking that enables \( t \).

The first requirement reflects the option to terminate. Eventual termination is not required; a workflow can be infinite, but an option to terminate should always remain open. The second requirement expresses the need in clear termination detection: A token in the sink place clearly specifies the termination of the workflow. The third requirement says that every transition should contribute to the work of the workflow—that is, is coverable (not dead).

Van Hee et al. [11] showed that if the first requirement is satisfied, then the second requirement is satisfied, too. The same property also holds for WFD-nets, which we introduce in the next section. For this reason, we do not consider the second requirement in this paper.

As we concentrate on conceptual workflow models, in which the enabling of a transition not only depends on the marking of the workflow net but also on the evaluation of a transition guard, it is possible that some transitions can become enabled in some executable versions of the conceptual workflow model and can never become enabled in others, which will be considered as expected behavior. At the same time, it is possible that for the correct functioning of the workflow some transitions should become enabled in all refinements of a conceptual workflow. This desire to have several levels of the third requirement (i.e., a subset of the transitions is coverable in at least one or in all refinements of a conceptual workflow model) led us to the adaption of the soundness definition from [11], in which soundness is equivalent to the option to complete (i.e., the first requirement), and the use of the notion of coverability to capture the third requirement.

**2.3. Adding data information to workflow nets**

A workflow net with data is a workflow net in which transitions (modeling tasks) can read from, write to, or delete data elements. A transition can have a (data dependent) guard that blocks its execution when it is evaluated to false. We assume a finite set \( D = \{d_1, \ldots, d_n\} \) of data elements, and we fix a set of predicates \( \Pi = \{p_1, \ldots, p_n\} \). We also assume a predicate labeling function \( \ell : \Pi \rightarrow 2^D \) that assigns to every predicate the set of data elements it depends on. When for a predicate \( p \in \Pi \), \( \ell(p) = \{d_1, \ldots, d_k\} \), we sometimes write \( p(d_1, \ldots, d_k) \) to emphasize this fact. A guard is constructed from predicates using the standard Boolean operations; the set of all guards over \( \Pi \) is denoted by \( \mathcal{G}_\Pi \). The function \( \ell \) naturally extends to guards, where \( \ell(g) \) for a guard \( g \) is defined as the union of \( \ell(p) \) over all the predicates \( p \) used in \( g \).

We now define a workflow net with data as a WF-net in which every transition \( t \) is annotated with at most four elements:

1. a set of data elements being read when firing \( t \),
2. a set of data elements being written when firing \( t \),
3. a set of data elements being deleted when firing \( t \),
4. a transition guard.

We do not explicitly consider the update of data elements, because this can be seen as the combination of read and write at the same transition. Furthermore, writing a data object for the first time models the creation of this data object.

**Definition 3 (WFD-net).** A workflow net with data (WFD-net) \( N = \langle P, T, F, rd, wt, del, grd \rangle \) consists of

- a WF-net \( \langle P, T, F \rangle \),
- a reading data labeling function \( rd : T \rightarrow 2^D \),
- a writing data labeling function \( wt : T \rightarrow 2^D \),
- a deleting data labeling function \( del : T \rightarrow 2^D \), and
- a guard function \( grd : T \rightarrow \mathcal{G}_\Pi \) assigning guards to transitions.

**Example 5.** An example of a WFD-net is the workflow of a shipper in Figure 1. Its data elements are \( D = \{cl, gds, ads, price, bon\} \), and its set of predicates is \( \Pi = \{\text{isHigh}(\text{price})\} \). Furthermore, \( \ell(\text{isHigh}(\text{price})) = \{\text{price}\} \). Consider transition calculate bonus. The labeling functions are \( rd(\text{calculate bonus}) = \{\text{price}\} \), \( wt(\text{calculate bonus}) = \{\text{bon}\} \), \( del(\text{calculate bonus}) = \emptyset \), and \( grd(\text{calculate bonus}) = \text{isHigh}(\text{price}) \).
The next section assigns formal semantics to WFD-nets.

3. Semantics of WFD-nets

The model of WFD-nets is a conceptual model, a schema for characterizing several executable workflows. In this section, we introduce a semantics for WFD-nets whose idea originates from the concept of hyper transition systems [7, 8] and allows us to capture all possible refinements of a WFD-net in one graph.

3.1. Behavior of WFD-nets

In a WFD-net, data values are not specified, but we still can distinguish non-created data values from created ones. Moreover, the value of a predicate can change only as a result of a change in the value of some data element the predicate depends on. In our semantics, we choose to keep the exact value for the predicates in a state. Predicates and guards can be evaluated to true, false, or undefined (if a data element assigned to them is undefined). We formalize this by three abstraction functions assigning abstract values to the data elements, the predicates, and the guards.

Function \( \sigma_D : \mathcal{D} \rightarrow \{\top, \bot\} \) assigns to each data element \( d \in \mathcal{D} \) either value \( \top \) (i.e., the value of the data element \( d \) is defined) or value \( \bot \) (i.e., the value of \( d \) is undefined). Function \( \sigma_{\Pi} : \Pi \rightarrow \{\top, \bot\} \) assigns to each predicate one of the values \( \top \) (true), \( \bot \) (false), and \( \perp \) (undefined). We impose a consistency requirement that \( \sigma_{\Pi}(\pi) = \top \) if and only if for a data element \( d \in \ell(\pi), \sigma_D(d) = \top \) (i.e., predicate \( \pi \) evaluates to true if an only if \( \pi \) depends on at least one undefined data element \( d \)). A pair \( \sigma = (\sigma_D, \sigma_{\Pi}) \) is a data state, and \( \Sigma \) denotes the set of all consistent data states. We use the following simplified notation: for data element \( d \in \mathcal{D} \), \( \sigma(d) = \sigma_D(d) \) and for predicate \( \pi \in \Pi \), \( \sigma(\pi) = \sigma_{\Pi}(\pi) \). As a guard \( g \in \mathcal{G}_{\Pi} \) is built from predicates using Boolean operations, \( \sigma(g) \in \{\top, \bot, \perp\} \) can be evaluated with function \( \sigma_{\Pi} \).

The next definition lifts the definition of a state of a WF-net to a WFD-net. We refer to a state of a WFD-net as a configuration.

Definition 4 (Configuration). Let \( N = (P, T, F, \text{rd}, \text{wt}, \text{del}, \text{grd}) \) be a WFD-net. A pair \( c = (m, \sigma) \) is a configuration of \( N \), if \( m \) is a marking of \( N \) and \( \sigma \in \Sigma \) is a consistent data state as previously defined. The start configuration of \( N \) is defined by \( ([\text{start}], ((d_1 \mapsto \top, \ldots, d_m \mapsto \top, \pi_1 \mapsto \bot, \ldots, \pi_n \mapsto \bot)) \). With \( \Xi \) we denote the set of all configurations, and \( \Xi_c = \{([\text{end}], \sigma) \mid \sigma \in \Sigma \} \) defines the set of final configurations.

In the initial configuration, only place start is marked; all data elements are undefined, and all predicates are evaluated to undefined. A configuration is a final configuration if it contains the final marking \([\text{end}]\).

Example 6. The initial configuration of the shipper in Figure 1 is defined to be \(([\text{start}], \sigma) \), where data state \( \sigma \) assigns \( \bot \) to all data elements \( \text{cl}, \text{gds}, \text{ads}, \text{price}, \text{bon} \). In addition, predicate \( \text{isHigh} (\text{price}) \) is \( \bot \). No data element is deleted in Figure 1, but we assume that upon reaching a final configuration (i.e., the case is completely executed), all data elements are deleted.

As Definition 4 lifts the notion of a marking (i.e., a state) of a WF-net to a configuration of a WFD-net, we have to define the behavior of a WFD-net in terms of configuration changes. For this purpose, we define the enabling condition for a transition \( t \) of a WFD-net \( N = (P, T, F, \text{rd}, \text{wt}, \text{del}, \text{grd}) \) in a configuration \( c = (m, \sigma) \) of \( N \).

The enabling condition of a transition \( t \) consists of two parts, both of which should be fulfilled. The first part takes the control flow into account and requires transition \( t \) to be enabled at marking \( m \). The second part considers the data state (i.e., the data values and the predicate values) in configuration \( c \). Any data element that is read by \( t \) or that is assigned to a predicate of the guard \( \text{grd}(t) \) must be defined. In addition, the guard of \( t \) must evaluate to true in this configuration.

An enabled transition \( t \) may fire. The firing of \( t \) may change the marking, the values of the data elements that have been written or deleted, and the values of predicates.

If the WFD-net is in configuration \( (m, \sigma) \), then the marking \( m' \) of a successor configuration \( (m', \sigma') \) is uniquely defined. It can be computed by firing transition \( t \) at marking \( m \) of the underlying WF-net \( (P, T, F) \) (i.e., \( m' = m - t + t' \)).

On the data level, we assume that inside a task reading always precedes writing and writing always precedes deleting. We assign undefined (i.e., \( \perp \) ) to each data element \( d \) that has been deleted when firing \( t \). Because writing always precedes deleting, no matter whether this data element has also been written, it is undefined after the firing of \( t \). In addition, we assign defined (i.e., \( \top \) ) to each data element \( d \) that has been written and not deleted when firing \( t \). All other data elements remain unchanged. Thus, also the values of data elements in a successor configuration are uniquely defined.

This is in contrast to the predicate values. As we have a conceptual workflow, the concrete data operations are not defined and we work with abstract data values. Therefore, the writing operations executed by transition \( t \) can result in more than one evaluation of the predicates, depending on the data that were written (in fact, depending on the concrete data operation that will be used later in the executable workflow)—we have to consider both, \( T \) and \( F \), as the possible successor value of a predicate \( \pi \) that depends on a data element \( d \) written by \( t \). More precisely, we assign undefined to each predicate that depends on one of the deleted data elements, and we (re)-evaluate each predicate that contains at least one such a data element \( d \) and no undefined data elements, considering both \( T \) and \( F \) options for each such a predicate.

As a consequence, the firing of \( t \) yields a set of successor configurations \( (m', \sigma') \). With our considerations, we conclude that each of these successor configurations has the same marking \( m' \), the same data values, and only the values of the predicates can differ. The following definition formalizes this.

---

1The meaning of the term configuration here is \( \"a state that includes data information\" \) and not the one related to configuring processes.
Definition 5 (Firing rule for WFD-nets). Let \( N = (P, T, F, rd, wt, del, grd) \) be a WFD-net. A transition \( t \in T \) of \( N \) is enabled at a configuration \( c = (m, \sigma) \) of \( N \) if \( m \xrightarrow{t} m' \), all data elements \( d \in rd(t) \) are defined, all data elements from \( \ell(grd(t)) \) (i.e., data elements assigned to the predicates occurring in the guard of \( t \)) are defined, and \( \sigma(grd(t)) = \top \) (i.e., the guard of \( t \) evaluates to true).

The firing of \( t \) yields a set \( C \subseteq \Xi \) of configurations with

\[
C = \{ (m', \sigma') \mid m \xrightarrow{t} m' \\
\land (\forall d \in del(t) : \sigma'(d) = \bot) \\
\land (\forall d \in wt(t) \setminus del(t) : \sigma'(d) = \top) \\
\land (\forall d \in D \setminus (del(t) \cup wt(t)) : \sigma'(d) = \sigma(d)) \\
\land (\forall \pi \in \Pi : (\forall d \in \ell(\pi) : \sigma'(d) = \top) \implies \pi' \in \{ \top, \bot \}) \\
\land (\ell(\pi) \cap wt(t) = \emptyset \implies \sigma'(\pi) = \sigma(\pi))\}
\]

and is denoted by \( c \xrightarrow{t} C \).

A configuration \( c \) of \( N \) is a deadlock if no transition \( t \in T \) is enabled at \( c \).

Example 7. Consider the WFD-net in Figure 1. The only transition, which is enabled at the initial configuration of this net, is receive goods whose firing results in the set of configurations including one single configuration with marking \( [p1] \), data elements \( cl, gds \) and \( ads \) being \( \top \), and data elements \( price \) and \( bon \) being \( \bot \). The value of predicate \( isHigh(\text{price}) \) in this configuration is \( \bot \), because the value of data element \( price \) is still undefined.

Transition calculate price is enabled at this configuration, because data element \( gds \) is defined. Transition calculate price takes data element \( gds \) as its input and stores its output in data element \( price \). The firing of this transition results in two configurations whose marking is \([p2, p3]\). As data element \( price \) was undefined before the occurrence of calculate price, a new value of \( price \) is created (otherwise, it would have been updated), and its value becomes \( \top \) in the new configurations. Moreover, as predicate \( isHigh(\text{price}) \) depends on the value of \( price \), this predicate is evaluated to either true or false, yielding two successor configurations. Note that in some refinements one of the two successor configurations can be unreachable; for instance, if the price could only be high, the \( isHigh(\text{price}) \) branch would be prohibited.

3.2. Reachability

Definition 5 defines the firing semantics for a single transition in a WFD-net. Now we extend the firing of a single transition to transition sequences and define the set of reachable configurations of \( N \). To take into account that we work with a conceptual workflow, and, therefore, we do not know what values the predicates will take in the executable workflow when the conceptual workflow is extended with the exact information about data operations and predicates, we define two notions of reachability: may-reachability and must-reachability. May-reachability guarantees that the reachability holds in at least one data refinement, whereas must-reachability guarantees that the reachability holds in every data refinement.

Given a configuration \( c \), a may-step from \( c \) specifies the possible existence of a successor configuration \( c' \) of \( c \) in at least one refinement. Accordingly, a may-path of length \( n \) specifies the existence of a sequence of \( n \) may-steps from \( c \) to a configuration \( c' \). In this case, we say that \( c' \) is may-reachable from \( c \). Any \( c \) is may-reachable from itself by the sequence of length 0.

A must-step from a configuration \( c \) specifies the set \( C \) of configurations that is reachable from \( c \) by firing a transition \( t \). We lift the notion of a must-step from a single configuration to a set \( C \) of configurations as follows: For each configuration \( c \in C \), a must-step is executed yielding a set \( C_c \). If no transition is enabled at \( c \) (i.e., \( c \) is a deadlock), we define \( C_c = \{ c \} \). Different transitions can be taken from different single configurations of \( C \). The union of sets \( C_c \) over \( c \in C \) forms \( C' \), and the set \( C' \) of configurations can be reached from a set \( C \) of configurations by executing a sequence of must-steps. In this case, we say that \( C' \) is must-reachable from \( C \).

Definition 6 (Reachability). Let \( N = (P, T, F, rd, wt, del, grd) \) be a WFD-net, \( c, c' \in \Xi \) be configurations of \( N \), and \( C, C' \subseteq \Xi \) be sets of configurations of \( N \).

- There is a must-step from a configuration \( c \) to a set \( C \) of configurations, denoted by \( c \xrightarrow{\text{must}} C \), if there is a transition \( t \in T \) enabled at \( c \) whose firing yields \( C \) (i.e., \( c \xrightarrow{t} C \)).
- There is a may-step from a configuration \( c \) to a configuration \( c' \), denoted by \( c \xrightarrow{\text{may}} c' \), if \( c' \) is an element of a set \( C \) of configurations that is reachable from \( c \) by a must-step (i.e., \( c \xrightarrow{\text{must}} C \land c' \in C \)).
- A may-path (of length \( n, n \geq 0 \)) from a configuration \( c \) is a sequence of configurations \( c_0, \ldots, c_n \) of \( N \) where for all \( 0 \leq i < n, c_0 = c \) and \( c_i \xrightarrow{\text{may}} c_{i+1} \); we denote the existence of a may-path \( c_0, \ldots, c_n \) with \( c_0 = c \) and \( c_n = c' \) by \( c \xrightarrow{\text{may}} c' \) and say that \( c' \) is may-reachable from \( c \).
- There is a must-step from a set \( C \) of configurations to a set \( C' \) of configurations, denoted by \( C \xrightarrow{\text{must}} C' \), if \( C' = \bigcup_{c \in C} C'_c \) for sets of configurations \( C'_c \subseteq \Xi \) such that either \( c \xrightarrow{\text{must}} C'_c \) or \( C'_c = \{ c \} \) if \( c \) is a deadlock.
- A must-path (of length \( n, n \geq 0 \)) from a set \( C \) of configurations is a sequence of sets \( C_0, \ldots, C_n \) of configurations of \( N \) where for all \( 0 \leq i < n, C_0 = C \) and \( C_i \xrightarrow{\text{must}} C_{i+1} \); we denote the existence of a must-path \( C_0, \ldots, C_n \) with \( C_0 = C \) and \( C_n = C' \) by \( C \xrightarrow{\text{must}} C' \) and say that \( C' \) is must-reachable from \( C \).

May-reachability always considers one successor and does not take into account multiple data refinements at the same time. In contrast, must-reachability combines successor configurations of \( c \) from different refinements. It thus opens a way for checking properties that would hold for all refinements.

Example 8. Figure 3 depicts the state space of the shipper. From the start configuration \( c_0 \), only the singleton set \( \{ c_1 \} \) can be reached by firing transition receive goods.
Configuration $c_1$ consists of a marking $[p_1]$, and the function $\sigma$ assigns the value $\top$ to data elements $c_1$, $gda$, and $ads$. Predicate $\text{isHigh}(\text{price})$ is undefined in $c_1$. Transition $\text{calculate price}$ is enabled at $c_1$. Firing this transition yields the set $\{c_2, c_3\}$ of successor configurations. The difference between both configurations is that in $c_2$ predicate $\text{isHigh}(\text{price})$ is evaluated to true (denoted $\text{isHigh}$), whereas in $c_3$ this predicate is evaluated to false (denoted $\neg \text{isHigh}$).

As $\{c_0\} \rightarrow_{\text{must}} c_1 \rightarrow_{\text{must}} \{c_2, c_3\}$, we conclude that there is a may-step from $c_0$ to $c_1$, from $c_1$ to $c_2$, and from $c_1$ to $c_3$. So there is a may-path of length 2 from $c_0$ to $c_2$ and from $c_0$ to $c_3$. Furthermore, $\{c_1\}$ but also $\{c_2, c_3\}$ is must-reachable from $\{c_0\}$. Observe that $\{c_1, c_1\}$ is also must-reachable from $\{c_0\}$. Whereas there is a must-step $c_1 \rightarrow_{\text{must}} c_1$, configuration $c_1$ does not have any successor. Hence, we conclude from the definition of must-reachability that $\{c_1, c_{12}\}$ is must-reachable from $\{c_0\}$ as well, and $c_{12}$ and $c_{13}$ are final configurations.

4. Executable workflow nets with data

With may- and must-reachability defined, we want to formalize the notions of may- and must-soundness for WFD-nets. As these notions are to be linked to the soundness/unsoundness of the executable workflows that can be obtained by refining a conceptual workflow, we first provide a (high-level) definition of an executable workflow being a refinement of a WFD-net. We shall use this definition to prove the correctness of our approach.

Executable workflow nets with data extend WFD-nets by providing concrete instructions on the computations that should take place for writing data and evaluating predicates. For the sake of simplicity, we ignore data types and assume $\Xi$ to be the set of all data values that includes the special element $\perp$—having the same meaning (i.e., undefined) as in WFD-nets—and values $T$ and $F$. We assume a set $\text{Expr}$ of (well-typed) expressions built over (constant) values and data elements. For an expression $e \in \text{Expr}$, $\text{var}(e)$ denotes the set of data elements appearing in $e$. The case when data is provided by the environment is captured by allowing an expression that results in producing a random number from a set of data values. We write $\text{BExpr}$ for the subset of expressions $\text{Expr}$ whose elements are valuated to Boolean values. Like in conceptual workflows defined by WFD-nets, we assume that whenever an expression is evaluated on a set of data elements one of which is undefined, the expression is evaluated to undefined as well.

Definition 7 (eWFD-net). An executable workflow net with data (eWFD-net) $N' = \langle P, T, F, rd, wt, del, grd, \text{cmpt}, \text{prexp} \rangle$ consists of

- a WFD-net $(P, T, F, rd, wt, del, grd)$;
- a data computation function $\text{cmpt} : T \rightarrow (\mathcal{D} \rightarrow \text{Expr})$ such that for every $t \in T$, the domain of the partial function $\text{cmpt}(t)$ is $\text{wt}(t)$ and for any $d \in \text{wt}(t)$, the data elements involved in the computation of the (new) value of $d$ belong to the set of data elements read by $t$; that is, $\text{var}(\text{cmpt}(t)(d)) \subseteq \text{rd}(t)$; and
- a predicate expression function $\text{prexp} : \Pi \rightarrow \text{BExpr}$ such that for every predicate $\pi \in \Pi$, the data elements involved in the computation of $\pi$'s value are declared to be involved in it in the WFD-net (i.e., $\text{var}(\text{prexp}(\pi)) \subseteq \ell(\pi)$), determining the way the predicates are defined.

Like the configurations of WFD-nets, configurations of eWFD-nets are pairs consisting of a marking and a data state. The set $\Xi'$ denotes the set of all consistent configurations of an eWFD-net. The semantics of eWFD-nets is defined in a straightforward way similarly to the may-semantics of WFD-nets, also implying that the data elements are undefined in the initial configuration. It differs from the WFD-net semantics only in the respect that concrete values are assigned to data elements, the computation operations are defined for writing data and evaluating predicates, and there is no distinction between may and must needed. The firing of a single transition $t$ from a configuration $c'$ leading to a configuration $c'_1$ in an eWFD-net is denoted by $c' \xrightarrow{t} c'_1$ and the reachability of $c'_1$ from $c'$ by $c' \xrightarrow{*} c'_1$.

Definition 8 (abstraction, refinement). For an eWFD-net $N' = \langle P, T, F, rd, wt, del, grd, \text{cmpt}, \text{prexp} \rangle$, the WFD-net $N = \langle P, T, F, rd, wt, del, grd \rangle$ is an abstraction of $N'$, and $N'$ is a refinement of $N$.

To formalize a link between the behavior of eWFD-nets and WFD-nets, we introduce the abstraction function $\alpha$ mapping the configurations of an eWFD-net $N' = \langle P, T, F, rd, wt, del, grd, \text{cmpt}, \text{prexp} \rangle$ to the configurations of the corresponding WFD-net $N = \langle P, T, F, rd, wt, del, grd \rangle$ as follows: For a configuration $c' = \langle m, s(c') \rangle$ of the executable workflow net $N'$, $\alpha(c') = \langle m, \sigma(c') \rangle$ with the same values for all the predicates—that is, for every $\pi \in \Pi$, $\sigma(\pi) = \sigma'(\pi)$—and the values of the data elements abstracting from the concrete data values, and only keeping track whether the data elements are defined or undefined—that is, for every $d \in \mathcal{D}$, $\sigma'(d) = \top$ if $\sigma'(d) = \bot$ and $\sigma'(d) = \top$ otherwise. The counterpart of $\alpha$, the concretization function $\gamma$, is defined as a mapping from the abstract configurations $c$ to the sets of configurations of the corresponding executable workflows whose abstraction is $c$—that is, $\gamma(c) = \{c' | \alpha(c') = c\}$.

It is easy to show that the defined functions $\alpha$ and $\gamma$ form a refinement relation between an eWFD-net $N'$ and the corresponding conceptual workflow $N$, similar to the refinement defined by Larsen and Thomsen [6]. That means, every must-step of the WFD-net covers some behavior of the eWFD-net, whereas every possible behavior of the eWFD net is captured by the may-paths of the corresponding WFD-net.

Theorem 1 (Refinement). Let $N'$ be an eWFD-net and $N$ the corresponding WFD-net. For every configuration $c \in \Xi$ and $c' \in \Xi'$ such that $c' = \alpha(c)$, the following two properties hold:

1. Every must-path is preserved through the refinement: If for some $C \subseteq \Xi$, $c \xrightarrow{\text{must}} C$ in $N$ then there exists a $c'_1 \in \Xi'$ such that $\alpha(c'_1) \in C$ and $c' \xrightarrow{*} c'_1$. 

2. Every step in the refinement has its counterpart (a may-step) in the abstract model: If for some \( c'_1 \in \Xi \), \( c' \xrightarrow{t} c'_1 \) then \( c \xrightarrow{\text{may}} \alpha(c'_1) \).

\[ \text{Proof. The proof follows immediately from Definition 8 together with Definition 5.} \]

The behavior of a WFD-net is even closer to the behavior of its refinements in the sense that every may-path in a WFD-net \( N \) can be reproduced in some refinement \( N' \) of \( N \).

**Theorem 2.** Let \( N \) be a WFD-net and \( c_0, \ldots, c_n \) be a may-path (of length \( n \)) from a configuration \( c_0 \) to a configuration \( c_n \). Then there exists an eWFD-net \( N' \) being a refinement of \( N \) such that for every \( 1 \leq i \leq n \), \( c'_0, \ldots, c'_n \) is a path in \( N' \) and \( \alpha(c'_i) = c_i \).

\[ \text{Proof.} \quad \text{We choose an} \ N' \text{in which every data element} \ d \text{takes values from the domain} \ D_d \text{of mappings from the set of the names of the predicates this data element depends on into the set} \{0, 1\}; \text{every predicate} \pi \text{is defined to be a sum modulo 2 over the values obtained for} \pi \text{over all the data elements from} \ell(\pi) \text{(i.e., the data elements} \pi \text{depends on), and the data computation function of a transition} \ t \text{is defined as writing a random value from} \ D_d \text{to a data element} \ d \in \text{wt}(t) \text{.} \text{\( N' \)} \text{clearly allows the required may-path of} \ N. \text{For the initial configurations, we have to choose} \ c'_0 \text{such that} \alpha(c'_0) = c_0 \text{implying that the markings and also the predicate values coincide. For every} \ c'_i \text{such that} \alpha(c'_i) = c_i, \text{we can simulate the may-step} \ c_i \xrightarrow{\text{may}} c_{i+1} \text{in} \ N \text{by choosing the firing} \ c'_i \xrightarrow{t} c'_{i+1} \text{of} \ t \text{which results in} c'_{i+1} \text{with the same values of the predicates as in} c_{i+1}. \text{Furthermore, the data element values are undefined in} \ N' \text{if and only if they are undefined in} \ N. \text{Thus,} \alpha(c'_{i+1}) = c_{i+1}. \]

**Lemma 1.** Let \( N \) be a WFD-net. For any set \( C \subseteq \Xi \) of configurations that is must-reachable from a configuration \( c_0 \in \Xi \)—that is, \( c_0 \xrightarrow{\ast \text{must}} C \)—and every \( c \in C \), we have \( c_0 \xrightarrow{\ast \text{may}} c \).

**Proof.** We prove the statement by induction over the length \( i \) of the must-path.

**Base** (\( i = 0 \)): By the definition of may- and must-reachability, the set of configurations reachable by the path of length 0 consists of the configuration \( c_0 \) only, which is may-reachable from \( c_0 \).

**Step:** Assume the statement holds for must-paths of length \( i \). For an arbitrary set \( C_{i+1} \) of configurations that is reachable from \( c_0 \) by a must-path of length \( i + 1 \), we have \( c_0 \xrightarrow{\ast \text{must}} C_i \xrightarrow{\ast \text{must}} C_{i+1} \).
with $C_i$ must-reachable from $c_0$ by a must-path of length $i$ (by Definition 6 (must-path)). Consider an arbitrary configuration $c_{i+1} \in C_{i+1}$. By Definition 6 (must-step), we must distinguish the following two cases: First, $c_{i+1} \in C_i$ (being a deadlock). By the induction hypothesis, $c_0 \rightarrow_{\text{may}} c_{i+1}$. Second, there is a $C_{i+1} \subseteq C_{i+1}$ containing $c_{i+1}$ and reachable by a must-step from some configuration $c_i \in C_i$: $c_i \rightarrow_{\text{must}} c_{i+1}$. By the definition of may- and must-steps (Definition 6), $c_i \rightarrow_{\text{may}} c_{i+1}$. By the induction hypothesis, $c_0 \rightarrow_{\text{may}} c_i$ from which we conclude, by Definition 6 (may-path), $c_0 \rightarrow_{\text{may}} c_{i+1}$.

**Theorem 3.** Let $N$ be a WFD-net. For any set $C \subseteq \Xi$ of configurations being must-reachable from the initial configuration (i.e., $c_0 \rightarrow_{\text{must}} C$) and every $c \in C$, there exist an eWFD-net $N' \subseteq N$ being a refinement of $N$ and a state $c'$ of $N'$ reachable from the initial state (i.e., $c_0' \rightarrow_{\text{may}} c'$) such that $\alpha(c') = c$.

Proof. Lemma 1 together with Theorem 2 directly imply the statement.

## 5. Soundness

In this section, we first define may- and must-soundness for WFD-nets and then prove the correctness of these notions. As usual, the must-property aims at the property that can be transferred to all the refinements; that is, must-soundness of a WFD-net guarantees that all executable workflow nets (i.e., WF-nets containing all data information) obtained by a data refinement from this WFD-net are sound. The may-property aims at guaranteeing the transfer of the negative verification result to all the refinements, meaning that if a WFD-net is not may-sound, all the executable workflow nets obtained from it by refining the data information are not sound. We go further than the standard may-must framework in [6] by showing additionally that (1) if a WFD-net is not must-sound, there exists an unsound refinement for this WFD-net, and (2) if a WFD-net is may-sound, there exists a sound refinement.

The set $\Sigma$ of all consistent data states is a finite set, because the number of data elements and predicates is finite and they take values from finite data domains. Moreover, for practical applications only WFD-nets with a finite set of reachable markings are of interest. This implies that the set $\Sigma$ of all configurations is finite as well. Therefore, we shall restrict our attention to WFD-nets with bounded underlying WF-nets and thus having a finite set $\Sigma$ of reachable configurations. As a consequence, an algorithm constructing all reachable configurations will eventually terminate.

In this section, we restrict the soundness property to the option to complete, the first requirement of the definition in [3]. Thus, soundness guarantees that from any reachable state it is always possible to reach a final state. The other way around, an unsound workflow contains at least one path that does not lead to a final state. Unsoundness is caused by the presence of a deadlock (i.e., a nonfinal configuration without successors) or a livelock (i.e., a set of nonfinal configurations that cannot be left). To capture these two faulty situations, we define the notions of a faulty state and a faulty set, which combine the two sources of unsoundness in the setting of some and all refinements.

First, we study when there exists an unsound refinement for a WFD-net $N$. An unsound refinement has a reachable state, say $c'$, from which it is not possible to reach a final state; that is, every path starting from $c'$ leads to a deadlock or livelock. If such a state $c'$ exists, then there exists a may-path to the respective abstract configuration $\alpha(c')$ and we can continue this may-path such that a final configuration is never reached. Instead of checking for such a may-path in $N$, witnessing the existence of an unsound refinement, we show that not all refinements have the possibility to reach a final state. This is the case, if no set of final configurations is must-reachable from $\alpha(c')$; that is, every must-reachable set of configurations contains at least one nonfinal configuration indicating that some may-path did not terminate. We refer to $N$ as may-faulty if this property holds.

**Definition 9 (may-faulty).** A WFD-net $N$ is may-faulty if there exists a configuration $c \in \Xi$ that is may-reachable from the initial configuration and there is no set $C \subseteq \Xi_f$ of final configurations that is must-reachable from $c$.

**Example 9.** Figure 4 shows the state space of the WFD-net from Figure 2. We choose $c = c_0$ (the initial configuration is may-reachable from itself). There is only one must-path $[c_0] \rightarrow_{\text{must}} [c_1] \rightarrow_{\text{must}} [c_2, c_3] \rightarrow_{\text{must}} [c_1, c_4] \rightarrow_{\text{must}} [c_2, c_4] = [c_1, c_4]$ alternate. As both sets are not a subset of $\Xi_f$, we can conclude that this WFD-net is may-faulty.

In our motivation for the notion of may-faulty we argued that we can check whether a set of final configurations is must-reachable from a configuration $\alpha(c')$ rather than whether there exists a may-path from $\alpha(c')$ to a deadlock or livelock. The following lemma justifies this argument.

**Lemma 2.** Let $c \in \Xi$ be may-reachable from the initial configuration in a WFD-net $N$. If, for all refinements $N'$ of $N$ and all concrete configurations $c' \in \gamma(c)$, it is always possible to continue every path starting from $c'$ to a final state, then there exists a must-path in $N$ with $c \rightarrow_{\text{must}} C \subseteq \Xi_f$.

**Proof.** Let $c \in \Xi$ and $c' \in \gamma(c)$ be as defined. Assume that for all refinements $N'$ of $N$ and all concrete configurations $c' \in \gamma(c)$, every path starting from $c'$ can always be continued to a final state. We construct the following must-path.

First, we choose a transition $t$ and a refinement $N' \subseteq N$ such that there exists a path $c' \rightarrow t \cdots \rightarrow c_f \in \Xi_f$ in $N'$. By hypothesis, such an $N'$ exists. By Theorem 1(2), there exists a may-step $c \rightarrow_{\text{may}} c_1$, and configuration $c_1$ is by Definition 6 (may-step) an element of a set $C_1$ of configurations in $N$ with $c \rightarrow t C_1$. For all $c_1 \in C_1$, there exists a state $c'_1 \in \gamma(c_1)$ reachable from some $c' \in \gamma(c)$ (Theorem 3). As from any $c'$ it is always possible to reach a final state, we conclude that every state in $\gamma(c_1)$ can be continued to a final state and thus, by the definition of refinement, all configurations $c_1 \in C_1$ are neither a deadlock nor part of a livelock. With the same argument, we can extend the must-path to $C_1 \rightarrow_{\text{must}} \cdots \rightarrow_{\text{must}} C_n$. Our hypothesis guarantees that
for all $1 \leq i \leq n$, $c_i \in C_i$ and every state in $\gamma(c_i)$ can be continued to a final state. As $N$ has only finitely many must-reachable configurations, this construction will eventually terminate in a set $C_n$ of configurations of $N$ and by our hypothesis $C_n \subseteq \Xi_f$. □

With the previous lemma, we can show that the may-faulty property of a WFD-net is a necessary and sufficient criterion for the existence of an unsound refinement.

**Theorem 4.** A WFD-net $N$ is may-faulty if and only if there exists an unsound refinement $N'$ of $N$.

**Proof.** $\Rightarrow$: By contradiction: Suppose all the refinements $N'$ of $N$ are sound. We prove that $N$ is not may-faulty. Consider an arbitrary configuration $c \in \Xi$ of $N$ that is may-reachable from $c_0$. Because all the refinements are sound, every configuration $c' \in \gamma(c)$ can reach a final configuration $c'_f \in \Xi_f$. By Theorem 1(2), each path in any refinement $N'$ can be mapped to a may-path in $N$. Furthermore, every may-path starting from $c$ can be mapped to a path in some $N'$ (Theorem 2). Therefore, we can combine all may-paths starting from $c$ to a set of must-paths (follows from Definition 6), and we know that there is a mapping between these must-paths and paths in $N'$ (Theorem 3). Applying Lemma 2, the paths from these $c'$ to final configurations can be combined to build a must-path from $c$ to a subset of the set $\Xi_f$ of final configurations of $N$ and, therefore, $N$ is not may-faulty.

$\Leftarrow$: By contradiction: Assume $N$ is not may-faulty. We show that all refinements of $N$ are sound. Suppose the opposite holds; that is, there is an unsound refinement $N'$. Then for some reachable state $c' \in N'$, there is no path to a final state of $N'$. By hypothesis (i.e., $N$ is not may-faulty), there is a may-path from $c_0$ to $\alpha(c')$, and we can find a path from $\alpha(c')$ to a set $C$ of final configurations. By Theorem 1(2), $N'$ implements a firing sequence through the sets of the must-path to one of the final configurations of $C$, thus resulting in a contradiction. □

Theorem 4 establishes a relation between may-faulty and the existence of an unsound refinement of a WFD-net. It serves as a precise criterion to find errors that are present in some refinements.

To find the errors that are present in all refinements, we have to show that any refinement can reach a state from which no final state can be reached. As this requires to consider states of all refinements, we analyze the must transition system of a WFD-net $N$ (i.e., the transition system having the must-reachable sets of configurations of $N$ as its nodes and an edge from $C$ to $C'$ whenever $C \Rightarrow_{\text{must}} C'$). The idea is to reflect a situation in the must transition system in which all refinements either deadlock or livelock. To this end, we have to find a set of must-reachable sets of configurations from which no may-path to a final configuration exists. The latter property ensures that no refinement is prevented from this faulty situation. We refer to this set as a faulty-set. Technically, a faulty-set is a terminal strongly connected component in the must transition system.

**Definition 10 (must-faulty, faulty-set).** A WFD-net $N$ is must-faulty if there exists a set $C \subseteq 2^{\Xi_f}$ of sets of configurations that are must-reachable from the initial configuration and, for all $C_1, C_2 \subseteq \Xi_f$, $C_2$ is must-reachable from $C_1$ while no $C \not\subseteq C$ is must-reachable from $C_1$, and $C_1 \cap \Xi_f = \emptyset$. We refer to $C$ as a faulty-set.

If the faulty-set is a singleton set without self loop, then all refinements can deadlock. Otherwise, at least one refinement may livelock.

**Example 10.** Consider again the state space in Figure 4 and the refinement $\{c_0\} \rightarrow_{\text{must}} \{c_1\} \rightarrow_{\text{must}} \{c_2, c_3\} \rightarrow_{\text{must}} \{c_1, c_4\} \rightarrow_{\text{must}} \{c_2, c_4\} \rightarrow_{\text{must}} \{c_1, c_4\} \ldots$. The cycle in which $\{c_1, c_4\}$ and $\{c_2, c_4\}$ alternate is a terminal strongly connected component but it is not a faulty set, because final configuration $c_4$ is contained in both sets of configurations.

A must-faulty WFD-net does not have a sound refinement.

**Theorem 5.** A WFD-net $N$ is must-faulty if and only if no sound $e$WFD-net $N'$ is a refinement of $N$.

**Proof.** $\Rightarrow$: Let $N$ be must-faulty and $C$ be some set of configurations from a faulty-set $C$. By Theorem 3, every refinement can reach some configuration that corresponds to a configuration from the set $C$. Suppose there is a sound refinement $N''$ and a configuration $c''$ with $\alpha(c'') \in C$ is reachable in $N''$. Because $N''$ is sound, it is possible to reach some final state $c''_f$ from $c''$. By Definition 6 and Theorem 1(2), some set $C_1$ is must-reachable from $C$ such that $\alpha(c''_f) \in C_1$. Thus, $C_1 \cap \Xi_f \neq \emptyset$, contradicting the assumption that $C$ is a faulty-set. Hence, $N$ has no sound refinement.

$\Leftarrow$: Assume $N$ has no sound refinement. Then in every refinement $N''$, we can find a path from $c''_f$ to either a deadlock or a livelock. According to Theorem 1(2), each such path has a counterpart (a may-path) in $N$. By Definition 6, we can construct from these may-paths in $N$ a must-path from the initial configuration to a set $C$ of configurations such that, for all $c \in C$, all $c'' \in \gamma(c)$ cannot reach any final configuration; that is,
they are either a deadlock or they belong to a livelock. By Def-
nitions 5 and 6, no final configuration is may-reachable from
a configuration from which no final marking can be reached, re-
sulting in the unsoundness of all the refinements.

Must-soundness ensures that from any configuration \( c \) that is
may-reachable from the start configuration, a subset of the final
configurations is must-reachable. That means that from every
configuration that is reachable in any of the refinements of a
WFD-net, a final configuration can be reached (because must-
soundness ensures that all configurations that are must-reachable
from the start configuration, a subset of the final configurations
in all sound refinements of a conceptual workflow). Therefore, the
WFD-net is must-sound. Moreover, from each state being may-reachable
from \( c_0 \), a final configuration is must-reachable. Hence, we
conclude that the WFD-net of the shipper is also must-sound.
In other words, the soundness property holds in any data refine-
ment of the WFD-net in Figure 1.

Let us now come back to the modification of the shipper (see Section 1) in which different predicates \( \text{isHighLeft}(\text{price}) \) and \( \text{isHighRight}(\text{price}) \) are used in the left and the right
part of the shipper in Figure 1. In this case, \( c_2 \) corresponds
to a configuration in which both predicates are evaluated to
true, and \( c_3 \) corresponds to a configuration in which both
predicates are evaluated to false. In addition, \( c_1 \) has two
more successors, say \( c_2' \) and \( c_3' \), corresponding to configu-
rances in which \( \text{isHighLeft}(\text{price}) \) is evaluated to true and
\( \text{isHighRight}(\text{price}) \) to false, and vice versa. Thus, we have
\( c_1 \rightarrow_{\text{must}} \{c_2, c_3, c_2', c_3'\} \). In configuration \( c_2' \), transitions
expression and no bonus are enabled. Firing these transi-
tions yields a configuration, say \( c \), where the shipper is in
marking \([p_5, p_8]\). Configuration \( c \) is a deadlock, and it is may-
reachable from the start configuration. Hence, there does not
exist a final configuration that is must-reachable from \( c \), thus
proving that the modified shipper is not must-sound. However,
the modified shipper is may-sound: any must-reachable set of
configurations contain a configuration from which \( c_{13} \) is may-
reachable. Indeed, there exists a data refinement in which it
is always possible to reach a final configuration—for example,
the one in which the two predicates \( \text{isHighLeft}(\text{price}) \) and
\( \text{isHighRight}(\text{price}) \) always evaluate to the same value.

We have already shown that the WFD-net from Figure 2 is
may-faulty and by Corollary 2 not must-sound. Intuitively, we
can make a refinement in which the predicate \( \text{pred} \) will al-
ways be evaluated to false, and the process can never leave
the loop. On the other hand, in every set of configurations, which
is must-reachable from the initial one, there is a configuration
from which a final configuration is may-reachable. Therefore,
the WFD-net is may-sound.

6. Covering Transitions

In the previous section, we defined may- and must-soundness
for conceptual workflows to characterize the soundness prop-
erty for their eWFD-net refinements. Although soundness guar-
antees proper termination, a sound (properly terminating) work-
flow may still exhibit transitions that never become enabled. To
ensure that all the functionality of a workflow specified by a
WFD-net can potentially be used, we consider now the verifica-
tion of a property similar to the third requirement of soundness
as defined by Van der Aalst in [3]: coverability. We extend may-
soundness by a condition ensuring that transitions are coverable
in at least one sound refinement. Accordingly, we extend must-
soundness by a condition ensuring that transitions are coverable
in all sound refinements of a conceptual workflow.

The straightforward adaptation of the coverability require-
ment for eWFD-nets is to request that every transition in an
eWFD-net is coverable. It is obviously desirable to have a
similar notion for conceptual workflows. However, concep-
tual workflows may have data-dependent choices, thus making
it possible that not all transitions can become enabled in any refinement. For example, for each WFD-net with at least one predicate, there exists a refinement in which this predicate is defined to be false; thus, the corresponding transition cannot be enabled. The option to forbid a certain branch of behavior is desirable, because it enables us to configure conceptual workflows depending on the current business plan of the organization; for example, the predicate isHighLeft(price) in Figure 1 can be set to false to (temporarily) move to a low-budget market segment. This example shows that the coverability notion of soundness in [3] is too restrictive in the setting of conceptual workflows. We consider, therefore, a weaker condition and require only a subset of the transitions to be covered. Which transition should be coverable can then be specified by the process designer.

The control flow of a WFD-net $N$ is configured by the data values that are assigned to each data element. We refer to a transition $t$ of $N$ as may-coverable if there exists a sound data refinement of $N$ in which $t$ is coverable. For example, the shipper in Figure 1 should know that tasks express shipment and normal shipment can potentially be used. We emphasize that we are only interested in transition $t$ becoming active in a sound refinement—if $t$ was coverable only in unsound refinements, it would mean that $t$ is still not usable in practice.

For some transitions, we might wish to require a stronger property: they should be must-coverable; that is, they are coverable in any data refinement. For example, we might want a payment transition to be coverable in any refinement.

The following definition formalizes the notion of covering a transition in a conceptual workflow.

**Definition 12 (cover, coverable may-path).** Let $N = \langle P, T, F, rd, wt, del, grd \rangle$ be a WFD-net. WFD-net $N$ covers a transition $t \in T$ if there exists a may-path from the initial configuration to a configuration at which $t$ is enabled (i.e., $\exists c \in \Xi : c_0 \rightarrow_{\text{may}}^* c \land c_0 \rightarrow_{\text{may}}^* t$).

A may-path from the initial configuration of $N$ is $t$-coverable if it ends in a final configuration and contains the firing of transition $t$ (i.e., $\exists c \in \Xi, C \subseteq \Xi : c_0 \rightarrow_{\text{may}}^* c \rightarrow_{\text{must}}^* C : (\exists c' \in C : c' \rightarrow_{\text{may}}^* t \land c_0 \rightarrow_{\text{may}}^* t \in \Xi)$).

If there is a $t$-coverable may-path $c_0, \ldots, c_n$ in $N$, then, by Theorem 2, some refinement $N'$ of $N$ has a terminating path $c_0', \ldots, c_n'$ that covers $t$ and for $0 \leq i \leq n$, $\alpha(c_i') = c_i$. As such a refinement is not necessarily sound, we want to know whether there is a sound refinement in which $t$ is covered, we also require that no faulty-set is must-reachable from any configuration $c_0, \ldots, c_n$. The latter ensures that for every configuration along this path, there is always at least one properly terminating path in one of the refinements. Such a transition $t$ is may-coverable in $N$, and we show that the property of may-coverable transitions supports our intuition indeed.

For must-coverability, the intuitive idea is to find a must-reachable set $C'$ of configurations such that transition $t$ is enabled at every configuration $c' \in C'$. As every refinement of $N'$ of $N$ that has a state that can be mapped to some configuration $c' \in C'$, we can be sure that $N'$ covers $t$. However, the existence of such a set $C'$ in $N$ is only a sufficient but not a necessary criterion for $t$ being coverable in all refinements $N'$, because not all the configurations enabling $t$ and belonging to different refinements are combinable in one must-reachable set of configurations.

To overcome this problem, we search for a must-reachable set $C$ of configurations that contains some configurations at which $t$ is enabled. For the remaining configurations $c \in C$, we check whether a set $C_c$ of configurations is must-reachable from $c$ so that $t$ is enabled at some configurations of $C_c$ and continue doing so recursively. That way, we construct for each configuration of $C$ a set of configurations in which every configuration enables $t$. The union of all these sets $C_c$ yields set $C'$, and the union of the concretizations $\gamma(c)$ over $c \in C'$ covers all possible refinements of $N$.

This construction can be incorporated in the definition of must-path by introducing a stuttering step into it: Whereas some configurations would continue their execution, others would (temporarily) withdraw from progress. Compared with the definition of must-step (Definition 6), the stuttering step enables us to add the deadlock configurations to the set of successor configurations and, in addition, also nondeadlock configurations (i.e., only an arbitrary subset of the enabled configurations must be fired).

**Definition 13 (stuttering must-step).** Let $N = \langle P, T, F, rd, wt, del, grd \rangle$ be a WFD-net and $C, C' \subseteq \Xi$ be sets of configurations of $N$.

- There is a stuttering must-step from $C$ to $C'$, denoted by $C \rightarrow_{\text{smust}}^* C'$, if $C' = \bigcup_{c \in C} C'_c$ for sets of configurations $C'_c \subseteq \Xi$ such that either $c \rightarrow_{\text{must}} C'_c$ or $C'_c = \{c\}$.
- A stuttering must-path (of length $n, n \geq 0$) from $C$ to $C'$ is a sequence of sets $C^0, \ldots, C^n$ of configurations of $N$, where $C^0 = C, C^n = C'$ and for every $0 \leq i < n, C^i \rightarrow_{\text{smust}} C^{i+1}$. We denote the existence of a stuttering must-path from $C$ to $C'$ by $C \rightarrow_{\text{smust}} C'$ and say that $C'$ is stuttering must-reachable from $C$.

All the statements proven in Section 5 w.r.t. must-reachability remain applicable to the stuttering setting: The only place at which stuttering could disturb the results is the proof of the induction step of Lemma 1. This proof already includes a consideration of a weaker form of stuttering when taking into account deadlock configurations. By removing the remark about $c_{i+1}$ being a deadlock from this proof, we directly obtain the proof for the stuttering case. The soundness checks presented in Section 5 can also be performed in the stuttering setting. We still prefer to keep stuttering and nonstuttering definitions intact. As the stuttering setting may yield an exponentially blow-up of must-reachable sets of configurations, the nonstuttering setting allows for better performance.

With the notions of a $t$-coverable may-path and a stuttering must-step, we can define may- and must-coverability to distinguish between transitions that should be coverable in at least one sound data refinement or in any sound data refinement.
Definition 14 (may- and must-coverable). Let \( N = \langle P, T, F, rd, wt, del, grd \rangle \) be a WFD-net. A transition \( t \in T \) is

- may-coverable if there exists a \( t \)-coverable may-path \( c_0, \ldots, c_n \) of \( N \) such that for all \( 0 \leq i \leq n \), no faulty-set is must-reachable from \( c_i \).

- must-coverable if \( N \) is must-sound and there exists a set \( C \) of configurations such that it is stuttering-must-reachable from the initial configuration and all configurations \( c \in C \) enable transition \( t \).

Example 12. Consider the example from Figure 4. Transitions \( t_1 \) and \( t_2 \) are must-coverable, because there exist must-reachable (singleton) sets \{c1\} and \{c2\} at which the respective transitions are enabled. Transition \( t_3 \) is not must-coverable, because every stuttering must-step leads to a set of configurations that contains either configuration \( c_1 \) or \( c_2 \). However, \( t_3 \) is may-coverable: There exists a \( t_3 \)-coverable may-path \( c_0 \Rightarrow \text{may} c_1 \Rightarrow \text{may} c_3 \Rightarrow \text{may} c_4 \) and, additionally, no faulty-set is must-reachable from the configurations \( c_0, c_1, c_3, c_4 \). With similar arguments, we can see that also transition \( t_4 \) is only may-coverable.

In the WFD-net from Figure 1, transitions receive goods and calculate price are must-coverable; all other transitions are may-coverable.

The following two theorems justify the notions of may- and must-coverable. A transition is may-coverable in a WFD-net \( N \) if it can be enabled in some sound refinement of \( N \), and it is must-coverable in \( N \) if it can be enabled in all refinements of \( N \).

Theorem 6 (justification may-coverable). Let \( N = \langle P, T, F, rd, wt, del, grd \rangle \) be a WFD-net. A transition \( t \in T \) is may-coverable in \( N \) if and only if there exists a sound refinement \( N' \) of \( N \) in which \( t \) is covered.

Proof. \( \Rightarrow \): Assume that transition \( t \) is may-coverable in \( N \), and let \( v = c_0, \ldots, c_n \) be a \( t \)-coverable may-path in \( N \) with \( c_n \in \Xi_r \). By Theorem 2, there is at least one refinement \( N' \) that enables a \( t \)-coverable path \( v' = c_0', \ldots, c_n' \) such that for all \( 0 \leq i \leq n \), \( \alpha(c_i') = c_i \). Suppose all refinements \( N' \) containing \( v' \) are sound. Then in every such refinement \( N' \), there exists a reachable state \( c' \) that is either a deadlock or part of a livelock. Construct a projection of \( N \) to a WFD-net \( N' \) that restricts \( N \) to those configurations for which a mapping to a state in some \( N' \) exists. By the construction of \( N' \), path \( v' \) is a must-path in \( N' \). As any refinement \( N' \) does not have data-dependent choices, every configuration of \( v \) in \( N' \) enables either a transition of \( v \) (which is a must-step as mentioned previously) or a must-step (which is not part of \( v \)). By hypothesis, all \( N' \) are sound. Thus, with Corollary 1, \( N' \) contains a must-reachable faulty-set; that is, there is a configuration \( c \in \{c_0, \ldots, c_n\} \) such that a faulty-set is must-reachable from \( c \). This contradicts our assumption that \( N \) is may-coverable.

\( \Leftarrow \): Assume that there is a sound refinement \( N' \) that covers transition \( t \). Then, there is a path \( v' \) from the initial state of \( N' \) to the final state, and \( t \) is fired on this path. By Theorem 1(2), we have a corresponding may-path \( v \) in \( N \), and this path is a \( t \)-coverable may-path. For every configuration reachable from an arbitrary state \( v' \) of path \( v' \), it is always possible to reach a final state, because we assume \( N' \) to be sound. Thus, by Theorem 1, all sets of configurations that are must-reachable from \( \alpha(c') \) from \( v \) contain a configuration that corresponds to a configuration from which a final configuration is may-reachable, and thus they are not faulty-sets. We conclude that no faulty-set is must-reachable from any configuration of \( v \).

Theorem 7 (justification must-coverable). Let \( N = \langle P, T, F, rd, wt, del, grd \rangle \) be a WFD-net. A transition \( t \in T \) is must-coverable in \( N \) if and only if every refinement of \( N \) is sound and covers \( t \).

Proof. \( \Rightarrow \): Let \( t \) be must-coverable in \( N \); that is, \( N \) is must-sound and there is a set \( C \) of configurations that is reachable by a stuttering must-path from the initial configuration such that \( t \) is enabled at every configuration \( c \in C \). By Corollary 2, all refinements of \( N \) are sound. Let \( N' \) be an arbitrary refinement of \( N \). Because stuttering must-reachability is preserved through refinement (by Theorem 1 reformulated in the stuttering setting), \( N' \) has a reachable state \( c' \) such that \( \alpha(c') \in C \). As all \( c' \in C \) enable \( t \), and the markings and the predicate valuations of \( \alpha(c') \) and \( c' \) coincide, we conclude that \( c' \) enables \( t \) as well. Thus, \( N' \) covers \( t \).

\( \Leftarrow \): Let every refinement \( N' \) of \( N \) be sound; that is, \( N \) is must-sound. Because \( t \) is coverable in every refinement, in every \( N' \) there is a configuration \( c' \) reachable from the initial configuration, such that \( t \) is enabled at \( c' \). Construct the set \( C' \) of such configurations \( c' \) and the set \( C \) consisting of \( \alpha(c') \). Then there is a set of configurations \( C' \subseteq C \) which is stuttering-must-reachable in \( N \). Every \( \alpha(c') \) enables \( t \), because the markings and the predicate valuations of \( c' \) and \( \alpha(c') \) coincide (by the definition of \( \alpha \)), which completes the proof.

A trivial but practical extension of the notion of must-coverability of a transition is the notion of must-coverability of a set of transitions. For the shipment process in Figure 1 we can, for example, say that in every refinement at least one of the transitions inform by call and inform by mail is coverable, and the client will thus be informed about the shipment one way or the other. We capture this property in the following definition.

Definition 15 (must-coverable set). Let \( N = \langle P, T, F, rd, wt, del, grd \rangle \) be a WFD-net. A set of transition \( T' \subseteq T \) is must-coverable if \( N \) is must-sound and there exists a set \( C \) of configurations such that it is stuttering-must-reachable from the initial configuration and every configuration \( c \in C \) enables some transition \( t \in T' \).

To obtain the justification of this definition, adapt Theorem 7 to the sets of coverable transitions. In contrast, for lifting the notion of may-coverability to sets of transitions, we do not need to extend Definition 14.
7. Related work

In this paper, we introduced a model for conceptual workflows and techniques to analyze such models. To structure related work, we compare our model with related models, review data-dependent analysis of workflows, and link our analysis techniques to related techniques from the area of computer-aided verification.

7.1. Modeling conceptual workflows

We proposed WFD-nets to specify conceptual workflows\(^2\). They extend WF-nets, as introduced by Van der Aalst [3], with conceptual read/write/delete data operations. In addition, we can attach an arbitrary (data-dependent) guard to each transition of a WFD-net that blocks the execution of this transition when it is evaluated to false. The support of arbitrary guards generalizes our previous model from [4] which only allowed predicate-negation and conjunctions.

Like the model in [4], WFD-nets can be seen as an abstraction from notations deployed by popular modeling tools, like Protos of Pallas Athena, which is a business process modeling tool using a Petri-net-based modeling notation. Protos is used by more than 1,500 organizations in more than 20 countries and is the leading business process modeling tool in the Netherlands. As another example, the industrial language WS-BPEL provides an opaque construct that can in principle be used to specify partially defined business processes and hide internal operational details and data operations. By building on the classical formalism of Petri nets (more precisely, WF-nets), we keep our framework easily adaptable to many industrial and academic languages.

7.2. Data-dependent workflow analysis

Since the midnineties, many researchers have actively worked on the topic of workflow verification. This research has resulted in several notions of workflow correctness. Most established is the notion of classical soundness, as introduced by Van der Aalst [3], which we considered in this paper. Several other notions of soundness have been proposed by relaxing the notion of classical soundness. Van der Aalst et al. [10] review research on soundness and compare the different soundness notions.

Our notions of may-soundness and must-soundness base on the more relaxed soundness property, as introduced by Van Hee et al. [11]. In contrast to classical soundness, Van Hee et al. do not require that every transition of a WF-net must be coverable. In our notions of may- and must-coverable, we added a more relaxed coverability criterion as in [3]. We only required some transitions to be coverable. This notion of coverability has also been applied by Stahl and Wolf [12] for the verification of services.

The notion of relaxed soundness [13] for WF-nets ensures that for each transition \(t\) of a WF-net, there is a run that enables \(t\) and can be carried forward to the final state. On the one hand, may- and must-coverable are stricter than relaxed soundness, because at least one or all refinements are sound. A relaxed sound WF-net may deadlock or livelock. On the other hand, relaxed soundness guarantees that, for each transition \(t\), there exists a run to a final state. In our approach, this is only the case if \(t\) is in the set of may- and must-coverable transitions.

Although there exists a plethora of workflow correctness criteria, they almost exclusively focus on the control-flow aspect of workflows and ignore the also important data perspective. Only little work has actually been done to extend the existing results on control-flow verification to the setting with data. Sadiq et al. [14] first mentioned the importance of data-flow verification in workflow processes. They identify several possible errors in the data flow, but provide no means for checking these errors. Sun et al. [15] conceptualize the errors from [14] using UML diagrams and present verification algorithms. This work is further extended and generalized in [16]. Eshuis [17] uses model checking techniques to verify the control- and data-flow perspective of business workflows. In our previous work in [4, 5], we systematically presented anti-patterns for workflows as temporal logic formulae including data as a first class citizen. The language WS-BPEL [18] specifies a set of syntactically rules (called profile) to decide whether an executable process is a valid refinement of an abstract process. Mietzner et al. [19] focus on generating syntactically valid executable processes of an abstract BPEL process. König et al. [20] proposed a more general profile. These approaches are restricted to a set of rules which do not cover all possible refinements of an abstract process, we present algorithms to analyze the behavior of conceptual workflow models.

To sum up, all these works look at data from a conceptual perspective, in a similar fashion as we do in this paper (the operation of deleting data is typically not considered). However, unlike the methods of this paper, none of the mentioned papers guarantees that verified workflows will stay (in)correct when implemented on an information system. This is an important aspect, and we believe that our work is unique in this respect.

7.3. Verification of may/must transition systems

Another branch of related work is research in the area of formal methods and computer-aided verification. The final target of researchers working in this area is usually the verification of programs or systems, which may contain complex data coming from large or even infinite data domains. To cope with the complexity of the objects to be verified, one usually tries to verify a simpler (i.e., abstract) system. Many abstraction techniques, such as predicate abstractions [21, 22] and abstract interpretation [23–26], and abstraction methodologies, such as counterexample guided abstraction refinement (CEGAR) [27, 28], have been proposed and successfully applied.

Larsen and Thomsen [6] originally defined may/must transition systems, also referred to as modal transition systems. Since then, many variants of may/must transition systems have been proposed, for example [26, 29–31], and then found multiple applications there, for example [8, 32]. To increase the precision of may/must transition systems, must-transitions were replaced

\(^2\)WFD-nets have been originally introduced in [9], a conference-version of this paper.
by must-hyper transitions [7, 8, 33]. A must-hyper transition connects a single state to a set of successor states. Shohram and Grumberg [34] introduce may-hyper transitions.

To reason about may/must transition systems and their extensions with hyper transitions, desired properties of a model need to be described in a three-valued temporal logic; we refer to the work of Bruns and Godefroid [35], Huth et al. [30], and Godefroid and Jagadeesan [36]. The abstraction guarantees that if a property holds in the abstract system, then it holds in the concrete system. Accordingly, if the property does not hold in the abstract system, it should not hold in the concrete system. To deal with the third value, undefined, the abstract model is refined. Several refinement techniques have been proposed; for example, Shohram and Grumberg [8, 34] split abstract states, and [33, 37–39] suggest game-based frameworks to iteratively refine the abstract model until it can be decided whether the property holds or not. Another approach is known as generalized model checking and was introduced by Bruns and Godefroid [29] and Godefroid and Jagadeesan [36]. The generalized model checking problem asks whether there exists a refinement of the abstract model that satisfies (fulfills) a desired temporal logic property. Generalized model checking is more precise than the approaches in which abstract states are split or the game-based abstraction refinement frameworks at the expense of an increased complexity.

The first difference between the results presented in this paper and the works in the area of formal methods is that our techniques are restricted to verify soundness, whereas the mentioned abstraction refinement frameworks can be used to verify any property of temporal logics, such as LTL [40], CTL [41], or the μ-calculus [42]. Soundness with and without requiring coverability can be expressed as a property in the logic CTL.

As the main difference, all approaches on abstraction refinement aim at reasoning about the correctness of a concrete system by verifying an abstract version of this system. As a consequence, if a temporal logic property is undefined in the abstract system, then the abstraction is not precise enough and needs to be refined to verify or refute this property. The techniques used to stepwise refine an abstract system are the key ingredient of any of these frameworks. In contrast, our verification target is the verification of a conceptual model in such a way that the result tells us about the behavior of all refined (i.e., executable) models. As we start with a conceptual workflow rather than an executable workflow, our abstraction is precise; that is, we do not need to apply abstraction refinement.

We can formulate our verification target as a generalized model checking problem that can be solved using the algorithms presented in [29, 36]:

1. May-soundness coincides with the generalized model checking problem for a conceptual workflow and property soundness.
2. Must-soundness coincides with the generalized model checking problem for a conceptual workflow and property unsoundness.

If the first item returns true, the conceptual workflow is may-sound; if the second item returns true, the conceptual workflow is must-sound. However, if the first or the second item return false, then we cannot derive that the conceptual workflow is not may-sound and not must-sound, respectively, because generalized model checking checks only an implication. Our techniques, on the other hand, enable the most precision possible: If a WFD-net is not must-sound, we know there exists an unsound refinement for this WFD-net; and if a WFD-net is may-sound, we know there exists a sound refinement.

8. Conclusion

We have proposed workflow nets extended with data (WFD-nets) as a formalization of conceptual workflow models. We have presented algorithms to verify whether any refinement or all refinements for given WFD-net are sound. Soundness guarantees the proper termination of a WFD-net and that certain transitions are not dead. Our algorithms achieve maximal precision w.r.t. soundness, but it is possible to extend the results to cope with other definitions of soundness.

Our work is can be seen a cross-fertilization of design and modeling frameworks coming from the field of process-aware information systems (PAIS) [1], and verification and abstraction approaches developed in the area of formal methods.

To apply the proposed verification techniques, they must enable the verification of conceptual workflow models of industrial size. The state space of a WFD-net $N$ is in the size of $O(|S| \cdot 3^{\Pi} \cdot 2^{|\mathcal{D}|})$ where $|S|$ denotes the size of the state space of the underlying WF-net of $N$, $|\Pi|$ the number of predicates in $N$, and $|\mathcal{D}|$ the number of data elements of $N$. The use of hyper transitions makes the size of the model exponential in the number of states of $N$. We can formulate our verification target as a generalized model checking problem [36]. As a result, verification complexity is at least quadratic in the size of $N$. Despite the complexity, we believe that our techniques are tractable, because it has been observed that size of industrial models than the worst case complexity bound.

Another requirement for applying a verification technique is the quality of the diagnosis. Business analysts need detailed feedback indicating the cause for an error in the model rather than an answer “not may-sound.” Our algorithms check the existence of a faulty situation witnessing the violation of must-soundness and may-soundness. A witness trace leading to such a faulty situation can be reported (as is standard in model checking [43]). Deriving more elaborate diagnosis results—for example, suggestions on how an error can be repaired—is an open research question and a line of further research.

In ongoing work, we plan to implement the proposed verification techniques for WFD-nets in the process analysis and discovery framework ProM [44]. As ProM provides import functionality for many industrial process modeling languages, the integration of our implementation in ProM will enable us to achieve direct applicability of our framework to conceptual workflows from practice. With this implementation, we can validate our theoretical results and adjust them if necessary.

In addition to control flow and data flow, resources are another important ingredient of a workflow model. For this purpose, we plan to work on extending our model to formalize
the resource perspective and to adjust the presented verification techniques thereof. In addition, we want to apply our results to the verification of services—that is, reactive process models.

References

[39] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, MIT, Cam-
bridge, Massachusetts, 2000.