ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240461

AI for Declarative Processes: Representation, Mining, Synthesis

Marco Montali [a,*]

[a]Free University of Bozen-Bolzano, Italy
ORCID (Marco Montali): https://orcid.org/0000-0002-8021-3430

Abstract. The integration of business process management (BPM) with artificial intelligence (AI) is driving unprecedented advancements in the creation of trustworthy, intelligent information systems. On the one hand, BPM poses unconventional, relevant questions about processes and the event data produced during their execution. On the other hand, AI offers established techniques that must be adapted and further enhanced to address these questions effectively. This synergy is particularly impactful in the case of flexible processes, which are best represented using a declarative approach – emphasising the temporal constraints that must be respected by the process, instead of explicitly detailing all the acceptable flows of activities. In this article, we overview how automated reasoning and learning techniques for temporal logics on finite traces provide robust foundations for representing, mining, and synthesising declarative processes. We cover established results as well as frontier research in this area.

1 Introduction

Business process management (BPM) is a discipline at the intersection of operations management, computer science, and information systems, whose grand goal is to support managers, analysts, and domain experts in the design, deployment, enactment, and continuous improvement of operational, socio-technical processes within an organisation [25]. An (operational) *process* is a collection of related events, activities and decisions that involve a number of actors and objects, and that collectively lead to an outcome that is of value to the organisation and its customers.

Once the *identification* of a relevant process within the organisation is conducted, the so-called *business process lifecycle* [25] provides a general framework for managing and continuously improving the process, through the following iterative steps:

1. *Discovery* - the process is documented and modelled.
2. *Analysis* - the process model is qualitatively and quantitatively verified and validated, to identify issues before execution.
3. *(Re)design* - the model is changed and adapted to best respond to the issues identified during analysis.
4. *Implementation* - change management is applied within the organisation to enable the execution of the (re)designed process; in parallel, the IT infrastructure of the organisation is configured towards execution support and automation.
5. *Monitoring* - when the process is finally running, relevant data are collected and analysed to determine how well the process is being performed, and whether the resulting, exhibited behaviour indeed aligns to the desired one; this provides the basis for a new iteration of the cycle.

Traditionally, these steps involve a large effort from process analysts, who apply conceptual modelling techniques to manually model the processes during the discovery phase, and who conduct interviews and observe the organisation to extract relevant data during monitoring. This is however radically changing. In fact, thanks to the increasing digitalisation of contemporary organisations, virtually every activity executed within a process is nowadays recorded in an information system. The availability of the resulting digital event data enables a completely different interpretation of the BPM lifecycle, as witnessed by the field of *process mining* [68], which is witnessing a tremendous growth in academia and industry [45].

Process mining has the overarching goal to streamline and improve operational processes based on the factual event data recorded within the organisation. The starting point of a process mining project is an *event log* containing the traces of events arising from the execution of different process instances (or cases) of a process – such as the execution of an order-to-delivery process on distinct orders, or the application of a clinical guideline to different patients. Each event in the trace refers to a specific transition (like start or completion) of an activity in the process, indicating when it happened and, possibly, who was responsible for it, in addition to other domain-specific data attributes.

Given an event log, process mining answers key questions such as: how does the process look like in reality? Where are bottlenecks and points of friction located? Does the actual execution of the process lead to compliance violations and deviations? This is done through three major tasks:

- *(automated) discovery* – the synthesis of a process model from an event log, with the aim of reconstructing what is actually happening inside the organisation;
- *conformance checking* – the comparison of the actual behaviour contained in the event log and the expected behaviour described in a reference process model, to detect whether deviations exist and, if so, where do they manifest.
- *enrichment* – the augmentation of a process model with key information extracted from the event log, such as information related to performance and timing for unveiling bottlenecks and queues.

Notably, while an event log contains historical data on already completed process instances, process mining can also provide *operational decision support* [68] on running cases, answering questions related to prediction and recommendation, such as: when is a running

* Email: marco.montali@unibz.it.

case going to complete? Which are the most likely activities that will be executed next? If one intends to steer the execution towards a desired outcome, what are the recommended next steps?

The convergence of model-driven and data-driven techniques for process analysis, as well as the integration of organisational and IT practices in process management, is ultimately giving rise to *process science* as the *interdisciplinary study of socio-technical processes* [71]. From the IT point of view, process science calls for the realisation of trustworthy, intelligent information systems for process support, integrating the key aforementioned process mining tasks within the BPM lifecycle. This is why *Integrative AI* is an essential enabler for process science: we need solid techniques to *reason* on the dynamics of process models, *learn* knowledge from the data contained inside event logs, and conduct different *forms of inference* relating event data and process models.

On the one hand, process science poses unconventional, challenging questions to AI researchers interested in dynamic systems and their temporal evolution, and calls for integrating AI techniques covering the full spectrum of symbolic, connectionists, and neuro-symbolic AI. On the other hand, AI methods, algorithms and technologies need to be adapted and further developed to address these questions effectively. This is only the start: as recently outlined in a research manifesto on *AI-augmented BPM* [26], integrative AI has the potential of unleashing a new generation of process-aware information systems capable to dynamically unfold and adapt processes by exploring improvement opportunities, in autonomy and through continuous conversation with their human principals.

In this paper, we overview how infusing AI techniques within process science has driven unprecedented advancements in management of processes that inherently *flexibile* [62]. Flexibility calls for a high degree of freedom by the stakeholders responsible for executing the process, in deciding how to execute each and every case, depending on the current circumstances. It has been argued that flexible processes can be effectively represented using a *declarative approach*, which emphasises the temporal constraints that must be respected by the process, instead of explicitly detailing all the acceptable flows of activities [60, 57, 55, 43]. Considering that in a work process every execution is expected to eventually reach a final outcome, this naturally calls for adopting *temporal logics on finite traces* [19, 21] to describe declarative processes and their executions. We then recall how automated reasoning and learning techniques for such temporal logics provide robust foundations for representing, mining, and synthesising declarative processes.

We cover established results as well as frontier research in this area, showing a relevant setting where process science calls for genuine advancements in the foundational and applied aspects of AI.

The paper is structured as follows. In Section 2 we introduce flexible, declarative processes, with particular reference to the DECLARE approach [60, 55]. In Section 3 we recall temporal logics on finite traces and their automata-theoretic characterisation. In Section 4, we then overview how AI techniques related to such temporal logics have been effectively adopted and enhanced to represent, mine, and synthesise declarative processes. Finally, in Section 5 we describe advanced, cutting-edge topics that extend this declarative setting in several directions.

2 Declarative Process Specifications

The execution of a work process consists of a possibly unbounded, yet *finite sequence* of activities focussed on a specific instance, or *case*. The case is evolved starting from an initial state and leading to one among different possible final states. For example, in an order-to-pay process every execution refers to a specific order, starts from its creation and finally leads to its correct payment or cancellation.

The activities contained in a work process may be executed in many possible different ways. Different activity orderings, called *flows*, may reflect variability in the process (e.g., the way an order is managed depends on the customer type), as well as the need of tailoring the evolution of the case depending on the circumstances (e.g., the acceptance or rejection of an order changes the way it is consequently evolved).

If one just focusses on the process control-flow (stripping off other relevant perspectives such as that of data and resources), specifying a work process then calls for identifying which activities/tasks are relevant for the process, and how such building blocks can/must be ordered over time to obtain an acceptable execution. By assuming that executing an activity is atomic (i.e., instantaneous), a process is nothing else than a (usually infinite) set of finite traces over a fixed, finite set of Σ of activities. Infinity typically comes from the presence of loops in the process, where the number of iterations is determined at execution time and hence cannot be bounded a-priori.

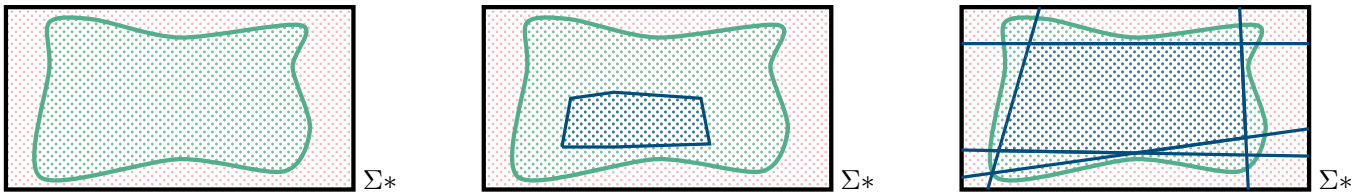
In this respect, *flexibility* refers to the ability of the process to support dynamic adaptations [62], delegating to the stakeholders responsible for the process execution to decide how to best streamline the evolution of each case. A maximally flexible process hence coincides with the set of all possible finite traces over Σ, usually denoted by Σ^*. However, not all such sequences represent valid, conforming executions. In fact, every process comes with *temporal/dynamic constraints* on its constituent activities, reflecting different forms of domain knowledge, such as hard/physical constraints (e.g., an empty order cannot be paid), legal/normative requirements (e.g., a shipment can only occur after the customer has signed consent for sharing their address), and best practices (e.g., an order is shipped only upon a prior, successful payment, as well as a priori confirmation of the warehouse). As shown in Figure 1(a), a process then singles out the subset of Σ^* representing which flows belong to the process. Modelling a process amounts to provide a compact, finite description of such a subset.

2.1 Modelling by Constraining

Standard process modelling approaches adopt flowchart-based notations (formalised via transition systems or Petri nets) that require to explicitly enumerate the flows of activities accepted by the process. They consequently struggle in covering the whole execution space when the process of interest is inherently flexible, and thus supports many distinct flows (see Figure 1(b)).

This motivated a the introduction of a different, *declarative* modelling style, providing a better balance between flexibility and control [15]. Specifically, declarative approaches support the direct elicitation of *what* are the relevant constraints on the temporal evolution of the process, without explicitly indicating *how* process instances should be routed to satisfy those constraints. This, in turn, calls for specifying as first-class citizens constraints dealing with *what is expected* to occur, as well as *what should not happen*. Every constraint separates traces from Σ^* into those that satisfy the constraint and those that violate it, and the entire specification then singles out those traces from Σ^* that satisfy all specified constraints. As highlighted in Figure 1(c), when the process is flexible, this approach promises to yield a better approximation of the boundaries of the real process, containing (and extending) those captured via procedural notations.

The idea of adopting a constraint-based, declarative approach to



(a) A process (b) Procedural specification (c) Declarative specification

Figure 1. Intuitive representation of the difference between procedural and declarative process modelling (inspired from [15]).

capture dynamic systems has been around for a long time in different communities. Seminal works include cascaded transactional updates in data management [18], declarative temporal specifications for agents [6] and their interaction protocols [66] in AI, and pockets of flexibility in BPM [65]. The idea of pockets of flexibility was further developed within BPM in consequent years, leading to a series of declarative, constraint-based approaches for the specification of processes, with DECLARE [59] and DCR Graphs [42] as main exponents. This led to a conceptual breakthrough in process science: while temporal constraints were before used to specify good/bad properties of a process, they were then employed to specify the process itself.

2.2 DECLARE

DECLARE [59, 57, 55] is a language and notation for the declarative specification of processes based on temporal constraints. An in-depth overview of DECLARE can be found in [15]; we provide here a brief account of its main features.

Every temporal constraint separates conforming traces that satisfy the constraint, from non-conforming traces that violate it. Instead of providing a full-fledged, logical language for specifying constraints of interest, DECLARE originally came with a catalog of pre-defined constraint templates, derived from a previously defined catalog of recurring temporal patterns used to specify properties of dynamic systems within software engineering [27]. While such properties were meant for dynamic systems generating traces of infinite length, in the context of DECLARE such patterns are interpreted over finite traces [15]. Further templates may be added depending on the modeller's needs (we come back to this point in Section 4.1).

Given a catalog of templates, a DECLARE specification S is a pair $\langle \Sigma, C \rangle$, where Σ is a finite set of activities, and C is a finite set of temporal constraints, each grounding one of the templates in the catalog on concrete activities from Σ. In the original catalog, templates are either unary, to constrain the number of occurrences of a single activity, or binary, to express temporal relations between the co-occurrence (or the absence thereof) of two activities. Constraints in C are interpreted conjunctively: a trace conforms to a DECLARE specification if and only if it satisfies all constraints in the specification.

A DECLARE specification can be rendered graphically, depicting each activity as a rectangle, each unary constraint as a decoration on the rectangle, and each binary constraint as an edge between pairs of activities, enriched with icons depending on its meaning [57].

Example 1. *Figure 2 graphically depicts a simple* DECLARE *specification tackling a (portion of) an order-to-pay process. The specification consists of four activities:* pick, close, pay, *and* cancel *– to respectively insert an item in the order, and complete/pay/cancel the order. The specification contains three constraints, instantiating the* precedence, response, *and* neg-succession *binary templates. Specifically,* precedence(pick,close) *indicates that the order can only be closed if* at least one *item has been picked* before. *Response*(close,pay) *dictates that every time the order is closed, it must be paid* afterwards. Neg-succession(cancel,close) *expresses a negative constraint: if the order is cancelled, it* cannot be *paid* afterwards.[1]

Notably, the resulting specification provides a wide degree of flexibility. For example, a completed order can be implicitly re-opened by adding further items; each payment, in turn, can cover one or multiple batches of items.

3 LTL on Finite and Process Traces

Linear Temporal Logic (LTL) [61] is a cornerstone modal logic for time. It augments propositional logic with modal operators referring to a discrete, linear flow of time. Specifically, the logic is interpreted over infinite sequences of states (called traces), where every state comes with a propositional valuation.

In a variety of application domains, including BPM (as discussed in Section 2), the system dynamics are more naturally captured using unbounded, yet *finite*, traces [21]. This consideration led to a logic called LTL *on finite traces* (LTL_f [19]), which adopts the syntax of LTL but interprets formulae on finite traces. Interestingly, moving from infinite to finite traces substantially impacts the semantics of the logic [21]. For example, while in LTL the two formulae $\neg \mathsf{X} a$ and $\mathsf{X} \neg a$ and are interchangeable, this is not the case in LTL_f. The first formula (which can be re-expressed as $\mathsf{X}_w \neg a$) in fact indicates that if a next instant exists, the activity occurring therein is not a. The second formula instead forces the existence of a next instant, in which a does not occur.

This radical semantical difference impacts also the automata-theoretic characterisation of the logic [19, 22]. In particular, every LTL_f formula φ can be encoded into a conventional *nondeterministic* finite-state automaton on finite words (NFA) that recognises all and only the traces that satisfy φ. Differently from the infinite-trace setting, every NFA can in turn be encoded into a corresponding *deterministic* finite-state automaton on finite words (DFA).

When dealing with declarative processes, we are specifically interested in a variant of LTL_f where propositions denote atomic activities constituting the basic building blocks of a process, and where each state indicates which atomic activity has been executed therein. As surveyed in [21, 15], this variant has been extensively used in AI and BPM. It has been then termed LTL *on process traces* (LTL_p) in [31], which contains a comprehensive investigation of the computational properties of the logic and its relationship with LTL_f. We adopt LTL_p throughout the paper, and recall it next.

Fix a finite set Σ of activities. A *(process) trace* τ over Σ is a finite sequence $a_0, \ldots, a_{n-1} \in \Sigma^*$, indicating which activity from Σ occurs in every instant $i \in \{0, \ldots, n-1\}$ of the trace. The length

[1] This is equivalent to say that an order can be closed only if no cancellation has occurred before [55].

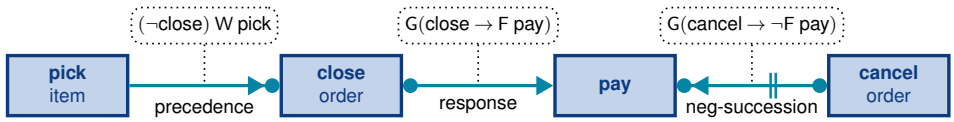


Figure 2. Fragment of an order-to-pay process in DECLARE, also reporting the LTL$_p$ encoding of its constraints.

n of τ is denoted $len(\tau)$, and $\tau(i)$ gets the i-th executed activity a_i in τ. As customary, Σ^* denotes the infinite set of all traces over Σ.

An LTL$_p$ formula φ is defined according to the grammar

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2, \text{ where } a \in \Sigma.$$

The semantics of LTL$_p$ is defined by the satisfaction relation $\models$.[2] Specifically, given an LTL$_p$ formula φ, a process trace τ, and an instance $i \in \{0, \ldots, len(\tau) - 1\}$, we inductively define that φ is true in instant i of τ, written $\tau, i \models \varphi$, as:

$$\begin{array}{ll}
\tau, i \models a & \text{if } \tau(i) = a \\
\tau, i \models \neg\varphi & \text{if } \tau, i \not\models \varphi \\
\tau, i \models \varphi_1 \vee \varphi_2 & \text{if } \tau, i \models \varphi_1 \text{ or } \tau, i \models \varphi_2 \\
\tau, i \models \mathsf{X}\varphi & \text{if } i + 1 < len(\tau) \text{ and } \tau, i + 1 \models \varphi \\
\tau, i \models \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{if } \tau, j \models \varphi_2 \text{ for some } j \text{ s.t. } i \leq j < len(\tau) \\
 & \text{and } \tau, k \models \varphi_1 \text{ for every } k \text{ s.t. } i \leq k < j
\end{array}$$

We say that τ satisfies φ, written $\tau \models \varphi$, if $\tau, 0 \models \varphi$. We say that φ is *satisfiable* if $\tau \models \varphi$ for some trace $\tau \in \Sigma^*$, and *valid* if $\tau \models \varphi$ for every trace $\tau \in \Sigma^*$.

Intuitively, X is the *(strong) next* operator: $\mathsf{X}\varphi$ indicates that the next instant must be within the trace, and φ is true therein. U is the *until* operator: $\varphi_1 \,\mathsf{U}\, \varphi_2$ indicates that φ_2 is true now or in a later instant j of the trace, and in every instant between the current one and j excluded, φ_1 is true.

The other boolean connectives are derived as follows: *(i)* $true = \bigvee_{a_i \in \Sigma} a_i$; *(ii)* $false = \neg true$; *(iii)* $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$; *(iv)* $\varphi_1 \rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$. In addition, other temporal operators are derived as follows: *(i)* $\mathsf{X}_w\varphi = \neg\mathsf{X}\neg\varphi$ – *weak next*, stating that if the next instant is within the trace, φ is true therein; *(ii)* $\mathsf{F}\varphi = true \,\mathsf{U}\, \varphi$ – *eventually*, stating that φ is true in the future; *(iii)* $\mathsf{G}\varphi = \neg\mathsf{F}\neg\varphi$ – *globally*, stating that φ is true in every future instant of the trace; *(iv)* $\varphi_1 \,\mathsf{W}\, \varphi_2 = (\varphi_1 \,\mathsf{U}\, \varphi_2) \vee \mathsf{G}\varphi_1$ – *weak until*, weakening *until* by allowing for the possibility that φ_2 does not become true in the future, and stating that in that case φ_1 will be true until the end of the trace. *(v)* $last = \mathsf{X}_w false$ – which holds true exactly in the *last* instant of the trace.

Example 2. *The top part of Figure 2 shows how the three* DECLARE *constraints from Example 1 can be encoded into* LTL$_p$.

Every LTL$_p$ formula can be encoded into a corresponding DFA over process traces - a standard DFA with the only difference that, due to how process traces are defined, it employs Σ instead of 2^Σ in labelling its transitions. A DFA over process traces from Σ is a tuple $A = \langle \Sigma, Q, q_0, \delta, F \rangle$, where: *(i)* Q is a finite set of states; *(ii)* $q_0 \in Q$ is the initial state; *(iii)* $\delta : Q \times \Sigma \rightarrow Q$ is the (Σ-labelled) transition function; *(iv)* $F \subseteq Q$ is the set of final states. A process trace $\tau = a_0, \ldots, a_n$ is accepted by A if there is a sequence of $n + 1$ states $q_0, \ldots, q_{n+1}$ such that: *(i)* the sequence starts from the initial state q_0 of A; *(ii)* the sequence culminates in a last state, that is, $q_{n+1} \in F$; *(iii)* for every $i \in \{0, \ldots, n\}$, we have $\delta(q_i, a_i) = q_{i+1}$. The language $\mathcal{L}(A)$ of A is the set of process traces accepted by A.

From well-known automata-theoretic constructions for LTL$_f$ [19, 22], which carry over LTL$_p$ with the notion of DFA defined above, we get that every LTL$_p$ formula φ can be encoded into a corresponding DFA A_φ, such that $\tau \in \mathcal{L}(A_\varphi)$ if and only if $\tau \models \varphi$ for every process trace $\tau \in \Sigma^*$.

While a few tools for reasoning on temporal logics on finite traces directly target LTL$_p$ [32, 14], the majority handles full LTL$_f$ (see, e.g., [73, 49, 70, 38, 36]). Two special LTL$_f$ formulae can be then used to force the reasoner to consider only process traces: *(i)* $\mathsf{G}\bigvee_{a_i \in \Sigma} a_i$ (to indicate that in each instant, some activity occurs), and *(ii)* $\mathsf{G}\bigwedge_{a_i, a_j \in \Sigma, a_i \neq a_j} \neg(a_i \wedge a_j)$ (to indicate that no two activities can occur in the same instant).

4 LTL$_f$-based Processes: Representation, Mining, Synthesis

The vision of process modelling by constraining (cf. Section 2.1), grounded in the DECLARE approach (cf. Section { sec:declare }), calls for: *(i)* a formal basis to unambiguously characterise which traces belong to a process, *(ii)* algorithmic techniques to tackle reasoning and analysis tasks from the BPM lifecycle. As we show in this section, LTL$_f$ and LTL$_p$ provide a very natural basis to tackle these two challenges.

4.1 *Representing* DECLARE *in* LTL$_p$/LTL$_f$

In the original catalog of temporal patterns from which DECLARE was initially derived [27], every pattern comes with a corresponding LTL representation. This has been indeed exploited, since the very inception of DECLARE [59, 57], to define the semantics of DECLARE itself. Notably, some formulae from the original catalog must be reformulated, to reflect the change from infinite to finite traces [21].

More specifically, every DECLARE template comes with a corresponding LTL$_p$ template formula. For example, template response(a, b) is represented as $\mathsf{G}(a \rightarrow \mathsf{F}b)$, where a and b are activity placeholders. The application of a template to concrete activities is then represented as a corresponding *constraint formula*, obtained by taking the template formula and replacing the placeholders with the corresponding activities (see, e.g., the case of response(close,pay) in Figure 2). An entire DECLARE specification $S = \langle \Sigma, C \rangle$ is then represented by the LTL$_p$ *specification formula* $\Phi_S = \bigwedge_{c_i \in C} \varphi_{c_i}$, where φ_{c_i} denotes the constraint formula of c_i.

Formula Φ_S formally characterises the set of traces that belong (or conform) to S (that is, the blue-dotted polygon in Figure 1(c)): a trace τ belongs to S if and only if $\tau \models \Phi_S$.

Example 3. *Consider the* DECLARE *specification in Figure 2, and the following four traces:*

- $\tau_1 =$ pick
- $\tau_2 =$ pick, pick, close
- $\tau_3 =$ pick, pick, close, pick, pay
- $\tau_4 =$ pick, pick, close, cancel, pick, pay

[2] For simplicity, we do not address next the corner case where the trace is empty. This case is tackled in detail in [22].

Traces τ_1 and τ_3 belong to the specification, while τ_2 and τ_4 do not.

Having an LTL_p-based representation of DECLARE also allows the modeller to introduce further LTL_p/LTL_f constraints, going beyond what supported in the core template catalog. More in general, the idea at the basis of DECLARE can be lifted to full-fledged LTL_p/LTL_f*-based processes*, specified using arbitrarily big conjunctions of "small" formulae, each capturing a temporal constraint.

4.2 Verification and Reactive Synthesis

While the semantics of a single, DECLARE constraint in isolation is quite intuitive to understand, the situation changes when focussing on an entire specification. On the one hand, temporal logics are notoriously hard to interpret for domain experts, even more in a finite-trace setting [21, 40]. On the other hand, conjoining temporal constraints has the effect of introducing so-called *hidden dependencies*, that is, further implicit constraints [57]. This makes it tricky to understand which traces indeed belong to the specification, and which not [41].

Example 4. *Consider the* DECLARE *specification in Figure 2. The interplay between the* response *and* neg-succession *contained therein introduces a hidden dependency between the two activities* cancel *and* close*: whenever the order is cancelled, it cannot be closed afterwards. In fact, if one would later close the order, the two constraints would* conflict*: neg-succession would forbid a later payment, while response would require one.*

The LTL_p characterisation of DECLARE recalled in Section 4.1 is instrumental to mitigate such issues. In particular, tools for LTL_p/LTL_f satisfiability can be employed to *verify* a DECLARE specification S [57, 72, 31, 46], checking key properties such as: does S admit at least one trace? Does S contain "dead" activities that cannot be executed at all? Is a given temporal constraint implied by S? Does a trace of interest belong to S? Verification is essential in the analysis phase of the BPM lifecycle. In the case where the specification is unsatisfiable, that is, no trace conforms to it, identifying and dealing with (possibly minimal) subsets of constraints witnessing unsatisfiability is key towards root cause analysis and redesign [63, 17].

A further leap is needed in the very relevant setting where the process is enacted by multiple, independent agents, responsible for distinct activities and constraints in the process. This setting naturally connects to the LTL_f *reactive synthesis* problem [20]. In the simplest case where one agent can be controlled, while all the others homogeneously play together the role of an uncontrollable external environment, this problem amounts to build (if possible) an execution strategy for controller, ensuring the satisfaction of an LTL_f specification regardless of what environment does. This problem has been extensively investigated in AI and formal methods, as witnessed by the highly performing tools for LTL_f synthesis regularly participating to the SyntCOMP competition (https://www.syntcomp.org).

In [37], the reactive synthesis problem has been for the first time casted for LTL_f-based processes, using an assume-guarantee formulation that very well fits information systems and BPM. Notably, while the synthesis problem is in general 2EXPTIME-complete for LTL_f and LTL_p [20], it is shown in [37] that if one focusses only on the core catalog of DECLARE templates, it can be solved via a *singly exponential-time algorithm*.

4.3 Automated Discovery

As discussed in Section 1, automated discovery tackles a different form of synthesis: that of a process model from an event log. In the context of LTL_f-based processes, this problem has been studied in two distinct settings. The first, *discriminative discovery*, is reminiscent of a machine learning classification task. Here, the input event log is partitioned in two subsets, respectively containing "positive" and "negative" traces. The goal is to learn a declarative process specification accepting all the positive traces and rejecting all negative ones. Traditionally, this problem has been investigated in AI tackling general LTL_f formulae (or extensions thereof) and using criteria on the syntactic structure of the formulae to determine the "best" discriminating formula (see, e.g., [9, 64]). A parallel line has instead focussed on LTL_f-based processes, in particular DECLARE, primarily exploiting inductive logic programming [12, 44] and SAT-based techniques [13].

Discriminative discovery is hard to apply in process science, as the event log recording executions of the process does not readily come with a criterion for splitting its traces. This is why process mining traditionally tackles a very different discovery problem, known under the term of *specification mining* in the software engineering literature. In this setting, all traces in the log are considered as possible, "positive" executions, and the goal is to synthesise a process model that "well-describes" the traces contained therein. The intuitive of "describing well" is substantiated through metrics tackling key process dimensions such as fitness, simplicity, and generalisation [68].

In the context of DECLARE *mining*, this problem has been effectively tackled using the following schema (see the survey in [15]), implemented in the most efficient existing algorithms and tools [52, 23, 2]. First, suitable metrics to relate traces and temporal constraints are defined, capturing if, and *how interestingly* a trace satisfies a temporal constraint. For example, even though traces τ_1 and τ_3 from Example 3 both conform to the specification in Figure 2, τ_3 is more interesting than τ_1 for constraint response(close,pay), as it explicitly triggers the expectation to pay.[3] Second, efficient, brute-force approaches are employed, instantiating the core DECLARE templates on the activities in the log, and computing the corresponding metrics. Finally, the best-scoring constraints are retained, using LTL_p reasoning to handle incompatibilities and redundancies [16].

4.4 Conformance Checking and Monitoring

In the context of LTL_f-based processes, conformance checking amounts to check whether a trace satisfies all constraints contained in a reference specification, indicating which are violated [11]. This can be efficiently solved by simply verifying, in parallel and in isolation, each temporal constrained contained in the specification [15].

Two, more interesting, settings emerge from this simple formulation. The first is about obtaining more informative feedbacks on the reasons underlying non-conformance. In particular, *alignment-based conformance checking* [10] amounts to determine, in the case where an observed trace does not conform to the specification, *how close* the trace is to a conforming trace – typically using variants of Levenshtein distance to relate traces. Notably, automated planning techniques have been proven extremely effective in tackling alignment-based conformance checking for LTL_f-based processes [39].

A second setting deals with running, evolving traces, towards operational decision support. This is key to detect non-conforming situations as early as possible, allowing process stakeholders to react and intervene at runtime. In this respect, *anticipatory monitoring* [47] appears particularly appealing, as it aims at returning a fine-grained feedback on the satisfaction status of a temporal constraint in a running trace, and detect violations at the earliest moment possible.

[3] This relates to the notion of *vacuity* in model checking.

Example 5. *Consider trace τ_4 in Example 3. The earliest instant where a violation of the order-to-cash specification can be detected is 4, that is, the instant where the order is cancelled. In that instant, none of the constraints of the specification is permanently violated, but due to the conflict discussed in Example 4, from instant 4 onwards it is impossible to conclude the execution by satisfying all constraints at once. Equivalently, it is certain that at least one constraint will be permanently violated in the future.*

This form of anticipatory monitoring has been first introduced for LTL_f-based processes in [51, 50], and then extensively studied in [22], with two particularly interesting results. From the foundational point of view, it has been shown that every LTL_f formula can be subject to anticipatory monitoring (which is not the case in the infinite-trace case). From the practical point of view, minor adjustments to standard DFA-construction algorithms are enough to effectively construct monitors that are informative at two levels: that of each single temporal constraint, and that of the entire specification (the latter being essential to detect situations such as that in Example 5).

5 Frontier Topics

In Section 4, we have overviewed AI techniques for representing, synthesising, and mining LTL_f-based processes. We now briefly tackle some key frontier topics in this area.

5.1 Dealing with Uncertainty

In the paper, we have so far implicitly assumed that both the process specification and the event log are certain. However, this assumption is too strict in a number of application domains.

At the specification level, it may not be necessary to force that every execution trace must satisfy all constraints contained in the specification. Some tolerance may instead apply, making the boundaries of the specification less crisp. When uncertainty related to the satisfaction of a constraint can be quantitatively estimated (for example, relaxing the order-to-cash specification in Figure 2 by indicating that closed orders are paid in 90% of the cases), the specification should be interpreted in a *probabilistic* sense: every constraint has a certain probability to be satisfied or violated by some randomly selected trace. This setting has been tackled by first introducing a probabilistic variant of LTL_f [53], and then by exploiting a fragment of this logic to define a probabilistic version of DECLARE [54, 1]. In the resulting setting, combined reasoning on probabilities and time must be conducted, towards singling out the different possible scenarios depending on which constraints are satisfied and which not. Scenarios and their probabilities are in turn related to traces as well as whole event logs, yielding a full spectrum of probabilistic, declarative process mining techniques [1].

Uncertainty is also essential when dealing with event data. On the one hand, event logs are often not explicitly present inside the information systems of an organisation, but are obtained through laborious socio-technical procedures for event data extraction and integration. On the other hand, event data may be inherently incomplete and noisy, or derived from complex activity/event recognition pipelines. This calls for handling different types of uncertainty, covering *probabilistic* [58, 29, 33] as well as *fuzzy* [24] event data. Interestingly, dealing with fuzzy data provides a natural basis for neuro-symbolic conformance checking pipelines [24].

5.2 Data-Aware Constraints

Until now, we have considered only activities and their flows. While this control-flow perspective defines the backbone of the process, other equally important perspectives exist, such as, most prominently, that of *data*.

A first step towards data-aware LTL_f-based process specifications is to enrich events with *data attributes* (for example, recording the amount of money involved in a payment, or the destination address of a shipment). Such data attributes come with different data types, and thus call for lifting LTL_f to first-order variants dealing with multiple theories at once, i.e., so-called LTL_f *modulo theories* [34]. Constraints are in this way enriched with the ability of expressing conditions over data attributes (such as to indicate that credit card payments are only eligible for gold customers), or to relate data attributes over time (such as to describe that the quantity assigned to a data attribute can never decrease over time).

In this data-enriched setting, even basic forms of reasoning become immediately undecidable. In fact, undecidability already holds:

- with numerical data attributes with local conditions (i.e., conditions applied on time instants) expressing increment, decrement, and equality [30];
- with abstract data attributes compared for (in)equality by taking arbitrarily distant instants in the trace [8].

Substantial progress has been obtained in finding decidable fragments and reconstruct corresponding automata-based techniques to tackle key tasks within the BPM lifecycle, in particular conformance checking and monitoring. Some of these well-behaved fragments admit explicit, faithful finite-state abstractions [7, 3, 4], while dealing with richer fragments requires a delicate combination of automata- and model-theoretic results [30, 35].

There is a second, more complex level where data prominently come into play: that where *distinct process instances are mutually related*. In fact, the typical assumption that every instance is evolved in isolation is often too restrictive: it is common for processes to co-evolve multiple objects at once, objects connected to each other via one-to-many and many-to-many relations. This is, for example, what happens in complex order-to-delivery processes where the content of an order may be shipped through multiple packages, each containing items belonging to the same or different orders.

These so-called *object-centric processes* are gaining momentum in process science [69, 28], but pose a wide range of open questions related to their representation, mining, and synthesis. In a declarative setting, shifting from a single-case to an object-centric perspective essentially amounts to move from a setting where constraints are global, to one where constraints are *scoped* by objects and their relationships. An example of relationship-scoped constraint is: an order can only be paid if the customer *owning that* order has signed a GDPR consent before. An initial attempt to equip DECLARE with the ability of expressing this type of constraint has been proposed in [5], employing temporal description logics to combine the structural dimension of data objects and their relations, with the temporal dimension dealing with their evolution. Research on this fascinating setting is still at its infancy, with only a few preliminary results on discovery [48] and reasoning [5].

5.3 Process Framing in AI-Augmented BPM

The vision of *AI-augmented BPM* [26] comes with a wide spectrum of unexplored questions in the space of LTL_f-based processes. An *AI-augmented BPM system (ABPMS)* is defined in [26] as *a process-*

aware information system that relies on trustworthy AI technology to reason and act upon data, within a set of restrictions, with the aim to continuously adapt and improve a set of business processes with respect to one or more performance indicators.

The lifecycle of an ABPMS expands the classical one (recalled in Section 1) along two directions. First, the traditional lifecycle phases are continuously iterated, and infused with AI capabilities. Second, the lifecycle includes additional tasks that can only be realised with AI support, namely those of adaptation, explanation, and automatic continuous improvement.

One particularly relevant aspect when moving from BPM to AI-augmented BPM, is that process modelling is lifted to a more flexible notion of *process framing*. Framing aims at capturing the boundaries within which the executions of one or more processes of interest should be confined, in turn leading to key feature of *framed autonomy*: an ABPMS can autonomously decide how to progress the execution, as long as the boundaries imposed by the frame are respected.

This idea naturally matches the declarative approach described in this paper, making LTL_f-based processes a natural candidate for process framing [56]. However, a number of challenges have to be tackled. We mention three, which touch on timely topics in AI:

1. the need of moving from verification and synthesis as described in this paper, to more sophisticated forms of synthesis dealing with quantitative objectives and maximally permissive strategies;
2. the need of infusing conversational systems (such as in particular those based on large language models) with the ability to conduct the different reasoning and analysis tasks described here;
3. the need of constraining predictive and prescriptive process analytics techniques, currently tackled mainly using neural networks and simulation, to ensure that predictions and recommendations conform to the temporal constraints of the process.

6 Conclusions

We have discussed how the synergic interplay of business process management and integrative AI has led to a series of key advancements in the management of flexible processes. We have in particular overviewed:

1. how such processes can be captures using a declarative approach based on temporal constraints;
2. how linear temporal logics on finite trace provide a solid, underlying formal basis for such processes;
3. how corresponding integrative AI techniques have been effectively used to represent, mine, and synthesise these processes.

We have then discussed a number of frontier topics, mentioning progress and open questions.

All in all, we hope to have convinced the reader that process science is a relevant and interesting field of application for AI, and also triggers genuinely novel research relevant for AI as such.

References

[1] A. Alman, F. M. Maggi, M. Montali, and R. Peñaloza. Probabilistic declarative process mining. *Inf. Syst.*, 2012. To appear.

[2] A. Alman, C. Di Ciccio, F. M. Maggi, M. Montali, and H. van der Aa. Rum: Declarative process mining, distilled. In *Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings*, volume 12875 of *LNCS*, pages 23–29. Springer, 2021.

[3] A. Alman, F. M. Maggi, M. Montali, F. Patrizi, and A. Rivkin. Monitoring hybrid process specifications with conflict management: An automata-theoretic approach. *Artif. Intell. Medicine*, 139:102512, 2023.

[4] A. Alman, F. M. Maggi, M. Montali, F. Patrizi, and A. Rivkin. A framework for modeling, executing, and monitoring hybrid multi-process specifications with bounded global-local memory. *Inf. Syst.*, 119: 102271, 2023.

[5] A. Artale, A. Kovtunova, M. Montali, and W. M. P. van der Aalst. Modeling and reasoning over declarative data-aware processes with object-centric behavioral constraints. In T. T. Hildebrandt, B. F. van Dongen, M. Röglinger, and J. Mendling, editors, *Proc. of the 17th International Conference on Business Process Management (BPM 2019)*, volume 11675 of *Lecture Notes in Computer Science*, pages 139–156. Springer, 2019.

[6] H. Barringer, M. Fisher, D. M. Gabbay, G. Gough, and R. Owens. METATEM: an introduction. *Formal Asp. Comput.*, 7(5):533–549, 1995.

[7] G. Bergami, F. M. Maggi, A. Marrella, and M. Montali. Aligning data-aware declarative process models and event logs. In A. Polyvyanyy, M. T. Wynn, A. V. Looy, and M. Reichert, editors, *Proc. of the 19th International Conference on Business Process Management (BPM 2021)*, volume 12875 of *Lecture Notes in Computer Science*, pages 235–251. Springer, 2021.

[8] D. Calvanese, G. De Giacomo, M. Montali, and F. Patrizi. Verification and monitoring for first-order LTL with persistence-preserving quantification over finite and infinite traces. In *Proc. of IJCAI*, pages 2553–2560, 2022.

[9] A. Camacho and S. A. McIlraith. Learning interpretable models expressed in linear temporal logic. In *Proc. of ICAPS, pages = 621–630, publisher = AAAI Press, year = 2019,*.

[10] J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich. *Conformance Checking - Relating Processes and Models*. Springer, 2018.

[11] J. Carmona, B. F. van Dongen, and M. Weidlich. Conformance checking: Foundations, milestones and challenges. In van der Aalst and Carmona [67], pages 155–190. ISBN 978-3-031-08847-6.

[12] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari. Exploiting inductive logic programming techniques for declarative process mining. *Trans. Petri Nets Other Model. Concurr.*, 2:278–295, 2009.

[13] F. Chesani, C. D. Francescomarino, C. Ghidini, D. Loreti, F. M. Maggi, P. Mello, M. Montali, and S. Tessaris. Process discovery on deviant traces and other stranger things. *CoRR*, abs/2109.14883, 2021. URL https://arxiv.org/abs/2109.14883.

[14] F. Chiariello, V. Fionda, A. Ielo, and F. Ricca. A direct ASP encoding for declare. In *Proc. of PADL*, volume 14512 of *LNCS*, pages 116–133. Springer, 2024.

[15] C. D. Ciccio and M. Montali. Declarative process specifications: Reasoning, discovery, monitoring. In W. M. P. van der Aalst and J. Carmona, editors, *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 108–152. Springer, 2022.

[16] C. D. Ciccio, F. M. Maggi, M. Montali, and J. Mendling. Resolving inconsistencies and redundancies in declarative process models. *Inf. Syst.*, 64:425–446, 2017.

[17] C. Corea, I. Kuhlmann, M. Thimm, and J. Grant. Paraconsistent reasoning for inconsistency measurement in declarative process specifications. *Inf. Syst.*, 122:102347, 2024.

[18] H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic based modeling and analysis of workflows. In *PODS*, pages 25–33. ACM, 1998.

[19] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In F. Rossi, editor, *Proc. of IJCAI*, pages 854–860. IJCAI/AAAI, 2013.

[20] G. De Giacomo and M. Y. Vardi. Synthesis for LTL and LDL on finite traces. In Q. Yang and M. Wooldridge, editors, *IJCAI*, pages 1558–1564. AAAI Press, 2015. ISBN 978-1-57735-738-4. URL http://ijcai.org/Abstract/15/223.

[21] G. De Giacomo, R. De Masellis, and M. Montali. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In *Proc. of AAAI*, pages 1027–1033. AAAI Press, 2014.

[22] G. De Giacomo, R. De Masellis, F. M. Maggi, and M. Montali. Monitoring constraints and metaconstraints with temporal logics on finite traces. *ACM Trans. Softw. Eng. Methodol.*, 2022.

[23] C. Di Ciccio and M. Mecella. On the discovery of declarative control flows for artful processes. *ACM Trans. Manage. Inf. Syst.*, 5(4):24:1–24:37, 2015.

[24] I. Donadello, P. Felli, F. M. Maggi, M. Montali, and C. Innes. Conformance checking of fuzzy logs against declarative temporal specifications. In *Proc. of BPM*, LNCS. Springer, 2024.

[25] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013. ISBN 978-3-642-33143-5.

[26] M. Dumas, F. Fournier, L. Limonad, A. Marrella, M. Montali, J. Rehse, R. Accorsi, D. Calvanese, G. D. Giacomo, D. Fahland, A. Gal, M. L. Rosa, H. Völzer, and I. Weber. Ai-augmented business process man-

agement systems: A research manifesto. *ACM Trans. Manag. Inf. Syst.*, 14(1):11:1–11:19, 2023.

[27] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In B. W. Boehm, D. Garlan, and J. Kramer, editors, *ICSE*, pages 411–420. ACM, 1999. ISBN 1-58113-074-0.

[28] D. Fahland. Process mining over multiple behavioral dimensions with event knowledge graphs. In van der Aalst and Carmona [67], pages 274–319. ISBN 978-3-031-08847-6.

[29] P. Felli, A. Gianola, M. Montali, A. Rivkin, and S. Winkler. Multi-perspective conformance checking of uncertain process traces: An smt-based approach. *Eng. Appl. Artif. Intell.*, 126:106895, 2023.

[30] P. Felli, M. Montali, F. Patrizi, and S. Winkler. Monitoring arithmetic temporal properties on finite traces. In *Proc. of AAAI*, pages 6346–6354, 2023.

[31] V. Fionda and G. Greco. LTL on finite and process traces: Complexity results and a practical reasoner. *J. Artif. Intell. Res.*, 63:557–623, 2018.

[32] V. Fionda and G. Greco. LTL on finite and process traces: Complexity results and a practical reasoner. *J. Artif. Intell. Res.*, 63:557–623, 2018.

[33] A. Gal. Everything there is to know about stochastically known logs. In *Proc. of ICPM*, pages xvii–xxiii. IEEE, 2023.

[34] L. Geatti, A. Gianola, and N. Gigante. Linear temporal logic modulo theories over finite traces. In *Proc. of IJCAI*, pages 2641–2647. ijcai.org, 2022.

[35] L. Geatti, A. Gianola, N. Gigante, and S. Winkler. Decidable fragments of $\mathrm{ltl}_{\mathrm{f}}$ modulo theories. In *Proc. of ECAI*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 811–818. IOS Press, 2023.

[36] L. Geatti, N. Gigante, A. Montanari, and G. Venturato. SAT meets tableaux for linear temporal logic satisfiability. *J. Autom. Reason.*, 68 (2):6, 2024.

[37] L. Geatti, M. Montali, and A. Rivkin. Foundations of reactive synthesis for declarative process specifications. In *Proc. of AAAI*. AAAI Press, 2024.

[38] G. D. Giacomo and M. Favorito. Compositional approach to translate ltlf/ldlf into deterministic finite automata. In *Proc. of ICAPS*, pages 122–130. AAAI Press, 2021.

[39] G. D. Giacomo, F. M. Maggi, A. Marrella, and F. Patrizi. On the disruptive effectiveness of automated planning for ltl$f$-based trace alignment. In S. Singh and S. Markovitch, editors, *Proc. of AAAI*, pages 3555–3561. AAAI Press, 2017.

[40] B. Greenman, S. Prasad, S. Zhu, G. De Giacomo, S. Krishnamurthi, M. Montali, T. Nelson, M. Zizyte, and A. Di Stasio. Misconceptions in finite-trace and infinite-trace linear temporal logic. In *Proc. of FM'24), noeditor = André Platzer and Kristin-Yvonne Rozier, publisher = Springer, year = 2024, series = LNCS, note = To appear,*.

[41] C. Haisjackl, I. Barba, S. Zugal, P. Soffer, I. Hadar, M. Reichert, J. Pinggera, and B. Weber. Understanding declare models: strategies, pitfalls, empirical results. *Softw. Syst. Model.*, 15(2):325–352, 2016.

[42] T. T. Hildebrandt and R. R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *PLACES*, volume 69 of *EPTCS*, pages 59–73, 2010.

[43] T. T. Hildebrandt and R. R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In K. Honda and A. Mycroft, editors, *Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software (PLACES 2010)*, volume 69 of *EPTCS*, pages 59–73, 2010.

[44] A. Ielo, M. Law, V. Fionda, F. Ricca, G. D. Giacomo, and A. Russo. Towards ilp-based LTL f passive learning. In *Proc. of ILP*, volume 14363, pages 30–45. Springer, 2023.

[45] M. Kerremans, D. Sugden, and N. Duffy. Gartner magic quadrant for process mining platforms. Technical report, Gartner, 2024.

[46] M. Laghmouch, B. Depaire, N. Gigante, M. Jans, and M. Montali. Declare moges: Model generator and specializer. In *Demo Track of ICPM*, volume 3648 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[47] M. Leucker and C. Schallhart. A brief account of runtime verification. *J. Log. Algebraic Methods Program.*, 78(5):293–303, 2009. URL https://nodoi.org/10.1016/j.jlap.2008.08.004.

[48] G. Li, R. Medeiros de Carvalho, and W. M. P. van der Aalst. Automatic discovery of object-centric behavioral constraint models. In W. Abramowicz, editor, *Proc. of the 20th International Conference on Business Information Systems (BIS 201)*, volume 288 of *Lecture Notes in Business Information Processing*, pages 43–58. Springer, 2017.

[49] J. Li, S. Zhu, G. Pu, L. Zhang, and M. Y. Vardi. Sat-based explicit LTL reasoning and its application to satisfiability checking. *Formal Methods Syst. Des.*, 54(2):164–190, 2019.

[50] F. M. Maggi, M. Westergaard, M. Montali, and W. M. van der Aalst. Runtime verification of LTL-based declarative process models. In S. Khurshid and K. Sen, editors, *RV*, volume 7186 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2011. ISBN 978-3-642-29859-2. URL http://dx.nodoi.org/10.1007/978-3-642-29860-8.

[51] F. M. Maggi, M. Westergaard, M. Montali, and W. M. P. van der Aalst. Runtime verification of ltl-based declarative process models. In *Proc. of the 2nd International Conference on Runtime (RV)*, volume 7186 of *LNCS*, pages 131–146. Springer, 2011.

[52] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In J. Ralyté, X. Franch, S. Brinkkemper, and S. Wrycza, editors, *CAiSE*, volume 7328 of *Lecture Notes in Computer Science*, pages 270–285. Springer, 2012. ISBN 978-3-642-31094-2. URL http://dx.nodoi.org/10.1007/978-3-642-31095-9.

[53] F. M. Maggi, M. Montali, and R. Peñaloza. Temporal logics over finite traces with uncertainty. In *Proc. of the 34 AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 10218–10225. AAAI Press, 2020.

[54] F. M. Maggi, M. Montali, R. Peñaloza, and A. Alman. Extending temporal business constraints with uncertainty. In D. Fahland, C. Ghidini, J. Becker, and M. Dumas, editors, *Proc. of the 18th International Conference on Business Process Management (BPM 2020)*, volume 12168 of *Lecture Notes in Computer Science*, pages 35–54. Springer, 2020.

[55] M. Montali. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach*, volume 56 of *Lecture Notes in Business Information Processing*. Springer, 2010. ISBN 978-3-642-14537-7.

[56] M. Montali. Constraints for process framing in ai-augmented BPM. In *Proc. of BPM Workshops*, volume 460 of *LNBIP*, pages 5–12. Springer, 2022.

[57] M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative specification and verification of service choreographies. *TWEB*, 4(1), 2010. URL http://nodoi.acm.org/10.1145/1658373.1658376.

[58] M. Pegoraro, M. S. Uysal, and W. M. P. van der Aalst. Conformance checking over uncertain event data. *Inf. Syst.*, 102:101810, 2021.

[59] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. DECLARE: Full support for loosely-structured processes. In *EDOC*, pages 287–300, 2007.

[60] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. DECLARE: Full support for loosely-structured processes. In *EDOC*, pages 287–300. IEEE Computer Society, 2007.

[61] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[62] M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012.

[63] M. Roveri, C. D. Ciccio, C. Di Francescomarino, and C. Ghidini. Computing unsatisfiable cores for ltlf specifications. *J. Artif. Intell. Res.*, 80: 517–558, 2024.

[64] R. Roy, D. Fisman, and D. Neider. Learning interpretable models in the property specification language. In *Proc. of IJCAI*, pages 2213–2219. ijcai.org, 2020.

[65] S. W. Sadiq, W. Sadiq, and M. E. Orlowska. Pockets of flexibility in workflow specification. In *ER*, volume 2224 of *LNCS*, pages 513–526. Springer, 2001.

[66] M. P. Singh. Distributed enactment of multiagent workflows: Temporal logic for web service composition. In *AAMAS*, pages 907–914. ACM, 2003.

[67] W. M. van der Aalst and J. Carmona, editors. *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*. Springer, 2022. ISBN 978-3-031-08847-6.

[68] W. M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016. ISBN 978-3-662-49850-7.

[69] W. M. P. van der Aalst. Object-centric process mining: Dealing with divergence and convergence in event data. In *Proc. of SEFM*, volume 11724 of *LNCS*, pages 3–25. Springer, 2019.

[70] B. F. van Dongen, J. De Smedt, C. Di Ciccio, and J. Mendling. Conformance checking of mixed-paradigm process models. *Inf. Syst.*, 102: 101685, 2021.

[71] J. vom Brocke, W. M. P. van der Aalst, N. Berente, B. van Dongen, T. Grisold, W. Kremser, J. Mendling, B. T. Pentland, M. Roeglinger, M. Rosemann, and B. Weber. Process science: the interdisciplinary study of socio-technical change. *Process Sci*, 2(1), 2024.

[72] M. Westergaard. Better algorithms for analyzing and enacting declarative workflow languages using ltl. In S. Rinderle-Ma, F. Toumani, and K. Wolf, editors, *BPM*, volume 6896 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2011. ISBN 978-3-642-23058-5. URL http://dx.nodoi.org/10.1007/978-3-642-23059-2_10.

[73] S. Zhu, L. M. Tabajara, J. Li, G. Pu, and M. Y. Vardi. Symbolic ltlf synthesis. In *IJCAI*, pages 1362–1369, 2017.