Multi-model Monitoring Framework for Hybrid Process Specifications

Anti Alman[1(✉)], Fabrizio Maria Maggi[2], Marco Montali[2], Fabio Patrizi[3], and Andrey Rivkin[2]

[1] University of Tartu, Tartu, Estonia
anti.alman@ut.ee
[2] Free University of Bozen-Bolzano, Bolzano, Italy
{maggi,montali,andrey}@inf.unibz.it
[3] Sapienza University of Rome, Rome, Italy
patrizi@diag.uniroma1.it

Abstract. So far, business process monitoring approaches have mainly focused on monitoring executions with respect to a single process model. This setting aptly captures monolithic scenarios from domains in which all possible behaviors can be folded into a single model. However, this strategy cannot be applied to domains where multiple interacting (procedural) sub-processes work under additional (declarative) constraints. For example, in healthcare, co-morbid patients may be subject to multiple clinical pathways at once, in the presence of additional, general constraints capturing basic medical knowledge. To support monitoring of thus emerging hybrid specifications, we propose a Multi-Model Monitoring Framework. On the one hand, the framework allows for a hybrid representation of a process, using both procedural and declarative models. This admits more flexible process model design as domain experts can focus on specific procedures and domain constraints without needing to merge them into one single specification. On the other hand, the framework includes an automata-based monitoring technique to simultaneously account for multiple models within one execution while resolving conflicts caused by the interplay of such models. We describe the overall framework, report on a prototypical implementation of the monitoring technique, and demonstrate its feasibility with a healthcare scenario.

Keywords: Business process monitoring · Data Petri net · Declare · Automaton · Hybrid process

1 Introduction

A key functionality of any process-aware information system is *monitoring* [12]. Monitoring concerns the ability to detect, and therefore handle, deviations appearing in ongoing process instances and, in most cases, requires the expected behavior of the process to be specified in advance. Such specifications are commonly created in the form of procedural or declarative process models, depending on the scenario at hand. In general, procedural models are more suitable for

X. Franch et al. (Eds.): CAiSE 2022, LNCS 13295, pp. 319–335, 2022.
https://doi.org/10.1007/978-3-031-07472-1_19

relatively structured processes (e.g., an automated manufacturing line), while declarative models are more suitable for flexible and knowledge-intensive processes (e.g., the management of a natural disaster).

The majority of existing monitoring approaches rely on the assumption that the full knowledge-base (control-flow, decision rules, temporal aspects, etc.) required for monitoring each possible process instance can be embedded into a single process specification. While this is a reasonable assumption for processes with homogeneous behavior (i.e., either structured or knowledge-intensive), there are domains in which processes are characterized by a combination of several (independently defined) procedural sub-processes working under additional (declarative) domain constraints. A prime example of this can be found in the medical domain, where patients with co-morbidities may be subject to multiple clinical pathways at once (consisting of various medical actions) that can be enriched with additional, context-dependent constraints. Monitoring such multi-model scenarios becomes a challenging task as it requires not only to account for checking a process instance against multiple heterogeneous models, but also to provide mechanisms for handling the interplay of those models. For example, prescribing a certain treatment without taking into consideration patient's preexisting conditions may result in adverse effects causing a worsening of the patient's state. Ideally, these conflicts should be detected at early stages of the process execution and reported to health providers so as to avoid undesired treatment outcomes.

To address these problems, we present a Multi-Model Monitoring Framework for hybrid process specifications (M3 Framework). In this work, our focus lies on the lifecycle and conceptual features of the framework, while all related formal and algorithmic details are delegated to a technical report [1]. The framework consists of multiple phases, starting from knowledge elicitation prior to process execution, and ending with the successful completion of a scenario specific monitoring task. We outline the main steps of these phases and present a prototypical implementation of the corresponding multi-model monitoring approach. This approach can seamlessly handle the interplay of process specifications, detect inevitable violations in advance, and provide recommendations on the next course of action based on a violation cost model (either avoiding violations or, if that is not possible, minimizing the total cost of violations). The key benefits of the M3 Framework are (1) support for monitoring process executions with respect to multiple process specifications simultaneously, thus circumventing the need to embed the full knowledge-base required for the execution of every possible process instance into a single process specification, and (2) support for both procedural and declarative process specifications, thus circumventing the need to force declarative knowledge into procedural specifications and vice versa.

The rest of this paper is structured as follows. Section 2 provides an example scenario for the M3 Framework. Section 3 describes the process components used in this paper. Section 4 and Sect. 5 outline the main phases of the M3 Framework and the proposed monitoring approach respectively. Section 6 reports on initial monitoring experiments. Section 7 provides an overview of related work and Sect. 8 concludes the paper by outlining directions for future research.

2 Example Scenario

To motivate the need for our Multi-Model Monitoring Framework for hybrid process specifications, we use the healthcare scenario from [22] that shows interactions between seemingly disjoint clinical guidelines (CGs) and basic medical knowledge (BMK). In what follows, we provide a description of the scenario and abbreviations of activities used in the next sections.

The first CG that we consider defines a "default" process for patients with hip fractures. It begins with an initial assessment (`AP`) of the patient's condition, which is then followed by a decision on performing a surgery (`SD`). If the patient has a high fever, cough, or there are reasons to reconsider any immediate surgical intervention, then it may be decided to postpone the surgery (`PS`) and repeat the assessment after the blocking issues have been resolved. Otherwise, the patient is taken into a room where pre-surgery anesthesia (`preSA`) is delivered, and then into an operating room where the surgery is performed (`S`). After the surgery, the patient is prescribed with post-surgery analgesia (`postSA`) for pain relief as well as physiotherapy for mobilizing the hip (`M`). Both activities can be repeated until the patient feels better and can end the treatment (`HFend`).

The second CG provides a process for diagnosing and treating a potential chest infection. First, a patient with a suspected chest infection is prescribed a chest X-ray (`CXray`). By analyzing the X-ray images, the health provider decides whether there are serious signs of a chest infection that have to be urgently treated (`TCI`) or the suspicions were not corroborated (`noCI`). In the former case, the patient is immediately prescribed with an amoxicillin therapy (`AT`).

Usually, in medical practice, surgical interventions are avoided or postponed whenever the patient suffers from other conditions that can cause additional complications. For example, the general anesthesia required for performing a surgery cannot be performed together with an amoxicillin therapy. This is a common BMK rule that should be considered with any combination of CGs and the best way of doing so is to represent it as a global, declarative constraint, requiring that a decision to perform a surgery cannot coexist with an amoxicillin therapy, thus connecting two seemingly disconnected CGs. Similarly, a constraint can be used to specify that, if a patient has high body temperature or cough, then an X-ray check must be performed. Constraints can also be used to further specify the behavior of already existing CGs. For example, to prescribe that mobilization therapy should be delayed in case of leg pain after the hip surgery reported during a regular post-operational assessment (`postOA`).

Based on the first constraint, if the patient is diagnosed with a chest infection, but it is also decided to perform a hip surgery, then there is a conflict between that constraint and the two CGs. In this outlier, but possible situation there would be three alternatives: violate the first CG (by skipping the surgery and further treatment), violate the second CG (by not prescribing amoxicillin therapy), or violate the constraint (and prioritize the CGs).

Informing medical experts about the presence of a conflict is crucial to help them in assessing the current situation, weigh the implications of one choice over the others, and finally make an informed decision. These decisions can be further

supported by assigning a violation cost to each CG and BMK specification. In this case, we can assume that highest violation costs should be given to the constraint, as any surgical complications should be avoided, and to the second CG specification, since there are no alternatives to the antibiotic therapy.

The formal models used for representing CGs and BMK specifications and how to address the above conflicts are discussed in the following sections.

3 Process Components

In this section, we define representations of declarative and procedural data-aware process specifications by relying on data Petri nets (DPNs) [11,16] and a variant of multi-perspective DECLARE (MP-DECLARE) [5] respectively. Note that DPNs and MP-DECLARE can be both translated into an equivalent finite-state automaton by using data abstraction and automata construction techniques [9]. This will be leveraged in Sect. 5 to monitor hybrid specifications.

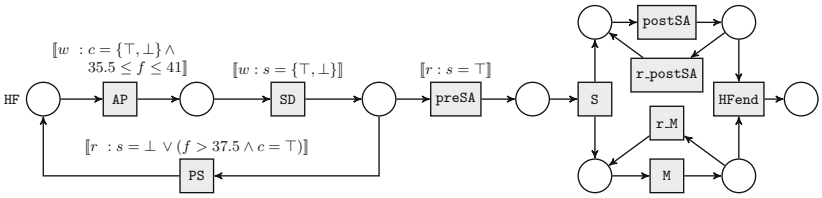
Events and Traces. An *event signature* is a tuple $\langle n, A\rangle$, where: n is the *activity* name and $A = \{a_1, \ldots, a_\ell\}$ is the set of event *attribute (names)*. We assume a finite set $\mathcal{E}$ of event signatures, each having a distinct name. By $\mathcal{N}_\mathcal{E}$, we denote the set of all event names from $\mathcal{E}$ and by $\mathcal{A}_\mathcal{E}$ the set of all attribute names occurring in $\mathcal{E}$.

An *event* of event signature $\langle n, A\rangle$ is a pair $e = \langle n, \nu\rangle$, s.t. $n \in \mathcal{N}_\mathcal{E}$, and $\nu : A \mapsto \mathbb{R}$ is a total function assigning actual values for the attributes of the corresponding signature. For simplicity, we assume attributes ranging over reals equipped with comparison predicates (simpler types such as strings with equality and booleans can be seamlessly encoded). As usual, we call a finite sequence $\sigma = e_1 \cdots e_\ell$ of events a *trace*, where each e_i is an event of some signature in $\mathcal{E}$.

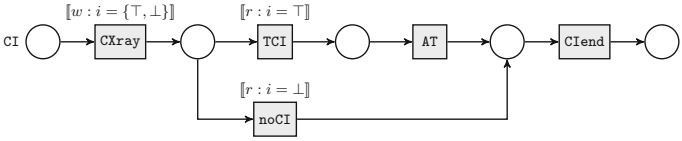
Data Petri Nets. As a language for procedural specifications, we opt for data Petri nets (DPNs) [11,16]. DPNs extend traditional place-transition nets with the possibility to manipulate scalar case variables, which are used as basic building blocks to constrain the evolution of the process through data-aware read-write constraints (called *guards*) assigned to transitions.

A *data Petri net* D over a set $\mathcal{E}$ of event signatures is a tuple $\langle P, T, F, l, V, r, w\rangle$, where: *(i)* (P, T, F) is the Petri net graph; *(ii)* $l : T \to \mathcal{N}_\mathcal{E} \cup \{\tau\}$ is a *labeling* function (here τ denotes a *silent* transition); *(iii)* $V \subseteq \mathcal{A}_\mathcal{E}$ is the set of net's *variables*; *(iv)* $r : T \to \mathcal{G}_\mathcal{E}$ (resp., $w : T \to \mathcal{G}_\mathcal{E}$) is a *read* (resp., *write*) *guard-assignment* function, mapping every transition $t \in T$ into a read (resp., write) guard – a boolean formula whose components are atomic expressions of the form $a \odot c$, where $\odot$ is a type-specific comparison predicate.

Example 1. Figure 1 shows two DPNs encoding the two CG fragments discussed in Sect. 2. Notice that the net in Fig. 1(a) models also a case in which the patient has a high fever and cough, but one of the doctors still recommends to risk performing the surgery (e.g., due to severe blood loss caused by the trauma). In this case, the model offers a choice: proceed with the surgery or postpone it (and possibly treat the blood loss together with fever and cough).



(a) DPN for hip fracture treatment



(b) DPN for chest infection treatment

Fig. 1. DPN representations for two clinical guideline fragments. We use prefixes *r:* and *w:* to distinguish read and write guards respectively. Models use the following CG parameters: coughs (c), fever (f), surgery decision (s), infection (i). Trivial, true-valued guards are omitted for brevity.

The execution semantics of DPNs extends that of place-transition nets with the ability to check and manipulate the net's variables via transition guards. In particular, a transition is *enabled* only if its read and write guards are satisfied under a given "firing mode" – a function that assigns values only to variables of the guards – and all the input places of the transition contain sufficiently many tokens to consume. Here, to check the read guard, the firing mode function picks values currently available in the net's state, while for the write guard it assigns to its variables any real values that would satisfy the guard (this accounts for "constrained" user input). When a transition is enabled, it may *fire* by consuming the necessary amount of tokens from its input places and producing the necessary amount of tokens in its output places, and by updating all the values assigned to variables in the write guard using the firing mode function. Values assigned to all other variables remain untouched.

In this paper, we deal only with DPNs that are 1-*bounded* and well-formed over their respective set of event signatures. The well-formedness means that a silent transition cannot update net variables and the write guard of each $t \in T$ uses variables matching attributes of event signature $\langle n, A \rangle$, for which $\ell(t) = n$. Such requirements appear naturally in the context of monitoring: we can always assume that variables are only manipulated by a fired visible transition, triggered by the corresponding event. At the same time, an event does not bring more data than is foreseen by the system design, and its payload is used to update the net variables (proviso that the corresponding write guard is satisfied).

Multi-perspective Declare with Local Conditions. To represent declarative process components, we opt for a multi-perspective extension of the well-known DECLARE language [5]. A DECLARE model describes *constraints* that

must be satisfied throughout the system (or process) execution. Constraints, in turn, are based on *templates*. Templates are patterns that define parameterized classes of properties, and constraints are their concrete instantiations. The syntax of such constraints is defined using the following grammar:

$$\Phi := \top \mid \varphi \mid \mathbf{X}\,\Phi \mid \mathbf{F}\,\Phi \mid \mathbf{G}\,\Phi \mid \Phi_1 \mathbf{U}\,\Phi_2 \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2$$

Here, φ is a boolean combination of attribute-to-constant comparisons and event variables ranging over $\mathcal{N}_{\mathcal{E}}$. Notice that the language of such boolean combinations without event variables closely resemble the guard language of DPNs, thus providing a good basis for combining declarative constraints with procedural models expressed with DPNs. As in standard LTL_f, $\mathbf{X}$ denotes the *strong next* operator (which requires the existence of a next state where the inner formula holds), while $\mathbf{U}$ stands for *strong until* (which requires the right-hand formula to eventually hold, forcing the left-hand formula to hold in all intermediate states).

We refer to the above constraints as LMP-Declare, i.e., MP-Declare constraints with local conditions. Their syntactic and semantic definition corresponds to LTL_f formulae [17] with attribute-to-constant comparisons and event variables as atomic formulae, interpreted over traces of the form given above.

Example 2. Here, we show how BMK constraints discussed in Sect. 2 can be formalized in LMP-Declare. The first constraint prohibits a positive surgery decision when the patient is undergoing amoxicillin therapy (Not Coexistence($\mathtt{AT}, \mathtt{SD}\{s = \top\}$)):

$$\big(\mathbf{F}(\mathtt{AT}) \rightarrow \neg\mathbf{F}(\mathtt{SD} \wedge s = \top)\big) \wedge \big(\mathbf{F}(\mathtt{SD} \wedge s = \top) \rightarrow \neg\mathbf{F}(\mathtt{AT})\big)$$

The second constraint stipulates that if a patient has high body temperature (i.e., greater than 37.5°) or cough, then an x-ray check for detecting a chest infection has to be performed (Response($\mathtt{AP}\{f > 37.5 \vee\ c = \top\}, \mathtt{CXray}$)):

$$\mathbf{G}\big(\mathtt{AP} \wedge (f > 37.5 \vee c = \top) \rightarrow \mathbf{F}(\mathtt{CXray})\big)$$

The third constraint prescribes that, whenever a patient suffers from pain in lower limbs (here modeled with variable *llp*) that is reported during a post-operational assessment, any mobilization therapy has to be delayed (Not Succession($\mathtt{postOA}\{llp = \top\}, \mathtt{M}$)):

$$\mathbf{G}\big((\mathtt{postOA} \wedge llp = \top) \rightarrow \neg\mathbf{F}(\mathtt{M})\big)$$

4 Multi-model Monitoring Framework for Hybrid Process Specifications

To address scenarios like the one described in Sect. 2, we introduce the Multi-Model Monitoring Framework (M3 Framework) for eliciting, managing, and monitoring hybrid process specifications. More specifically, the framework can be used in scenarios with multiple procedural and declarative specifications, where the former can be executed concurrently and the latter work as global constraints

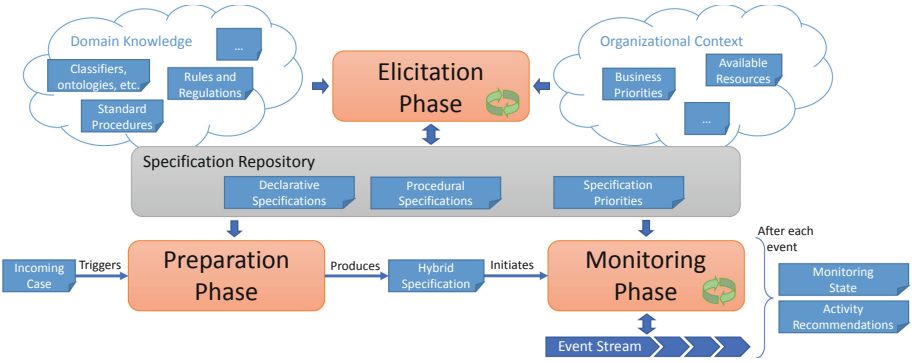


Fig. 2. Conceptual overview of the M3 Framework.

that implicitly induce additional dependencies between procedural specifications. In the following, we describe in more detail the three phases interconnected by a specification repository (Fig. 2), which is constantly updated during the model elicitation phase, and then used for monitoring individual cases in the case-specific preparation and monitoring phases.

4.1 Elicitation Phase

The elicitation phase of the M3 Framework is envisioned as a continuous case-agnostic phase, during which domain knowledge (standard procedures, classifiers, etc.) and organizational context (business priorities, available resources, etc.) are transformed into concrete process specifications, which are then stored in a dedicated specification repository. Both declarative and procedural specifications are supported, and multiple specifications of either paradigm can be used to represent a single global process, thus supporting not only hybrid process specifications, but also, for example, component-based and aspect-oriented modeling approaches [10]. Full agreement between the individual process specifications is not assumed, instead each specification is associated with a priority value that is used during monitoring to provide guidance to the user in resolving any potential conflicts. The elicitation phase consists of four main steps:

Extraction — Analysis of changes in the domain knowledge and organizational contexts to determine if new process specifications need to be created or existing ones need to be updated or removed. Relevant knowledge is extracted from standard procedures, regulations, etc., and the corresponding process specifications are identified.

Harmonization — Comparison of the extracted knowledge to common classifiers, ontologies, etc., to ensure that the same vocabulary is used across all process specifications. For example, specifications in the medical domain should adhere to the classifiers published by the World Health Organization.[1]

[1] Classifiers available at: https://www.who.int/standards/classifications.

Modeling — Transformation of the extracted knowledge into concrete process specifications via process modeling. Any modeling language (declarative or procedural) can be used, assuming that a translation into an equivalent finite-state automaton is possible (see Sect. 5.2 for more details).

Prioritization — Assignment of a priority value to each process specification. The priority value corresponds to the cost of violating the corresponding specification, and it is used during the monitoring phase (Sect. 4.3) to provide activity recommendations in case of conflicting specifications.

4.2 Preparation Phase

The preparation phase of the M3 Framework is envisioned as a case-specific non-recurrent phase in which an incoming case is assessed, relevant process specifications are selected, and a corresponding hybrid specification is automatically created as an input for the following monitoring phase. This hybrid specification will encompass the combined behavior of all selected specifications, the corresponding specification priorities (Sect. 4.1, Prioritization step), and, if required, also additional case-specific modifications. The selected process specifications can be smaller fragments of a single business process, but also fragments or full specifications of multiple, separately defined (but concurrently executed) business processes. The preparation phase consists of three main steps:

Case Assessment — Selection of relevant process specifications from the specification repository based on the features of the incoming case. In some domains, this step can be partially automatized. For example, in healthcare, potential conflicts between drugs used in different process specifications could be detected automatically using already existing knowledge bases.[2]

Fine-tuning — Optional modification of the selected specifications, including the specification priorities, and addition of custom specifications in order to tailor the monitor behavior to the specific needs of the incoming case. In healthcare, for example, this would include constraining the use of specific drugs or procedures because of some underlying conditions, co-morbidities, or preferences of the patient.

Automated Assimilation — Automated creation of a hybrid specification, which combines the outcome of case assessment and fine-tuning steps into a single case-specific hybrid specification, which also serves as the monitoring automaton. This step is further discussed in Sect. 5.2.

4.3 Monitoring Phase

The monitoring phase of the M3 Framework is envisioned as an ongoing case-specific phase, covering the entire duration of the case being monitored. During this phase, the state of the monitor and the set of next recommended actions

[2] For example: https://reference.medscape.com/drug-interactionchecker.

(with payloads) are updated after the occurrence of each event, therefore providing guidance towards the successful completion of the case. The monitoring phase consists of three main steps:

Automated Event Processing — Update of the monitor state, including both the global state and the states of individual process specifications, and update of the set of next recommended activities, including the corresponding data payloads. This step is further discussed in Sect. 5.3.

Case Reassessment — Optional modification of the current hybrid process specification to handle emerging case characteristics that were unforeseen during the preparation phase. For example, case reassessment would be necessary in the medical domain if a previously unknown underlying condition is discovered during the ongoing treatment case. Additionally, this step is useful for handling any imperfections that may be detected in the original process specifications during the ongoing monitoring case. Case reassessment is currently unsupported in the preliminary implementation of the monitor.

Next Event Execution — Execution of the next event based on the current monitoring state and activity recommendations. While we see recommendations as an important input for guiding the process execution to a successful completion, the execution of other (not recommended) events is also possible.

5 Automata-Based Monitoring

There are two central challenges in the proposed M3 Framework. First, the assimilation of input process specifications into a single hybrid specification (required for Sect. 4.2). Second, the interpretation of incoming events against this hybrid specification (required for Sect. 4.3). From an algorithmic perspective, these challenges can be addressed by encoding each DPN and LMP-Declare specification into a corresponding guarded finite-state automaton (GFA) that fully captures the execution semantics of the specification. This is possible, even in the presence of data, thanks to the specific shape of data guards (Sect. 3), which allows the usual automata constructions to be augmented with data abstraction and consequent propositionalization techniques [3]. In the following, we give a conceptual overview of the necessary steps, while formal and algorithmic details are delegated to the technical report [1].

5.1 Monitoring Semantics

A prerequisite of any monitoring approach, especially in a novel hybrid setting such as the one induced by the M3 Framework, is to set the corresponding monitoring semantics. At the level of individual process specifications, the standard execution semantics of data Petri nets or LMP-Declare constraints are followed. However, the execution semantics of a hybrid specification, produced in the Preparation phase (Sect. 4.2), is specific to the M3 Framework and therefore warrants further discussion. The following rules apply:

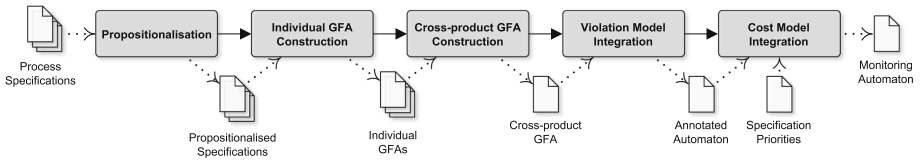


Fig. 3. Main steps and intermediate artefacts for creating the monitoring automaton.

- Each DPN ignores events that are not included in that DPN. This is a natural consequence of modularity which requires not to assume that a single DPN explicitly describes the full control flow (and therefore all potential activities) of the global process. Note that this rule does not apply to LMP-Declare, since declarative constraints are interpreted globally. In particular, that would break the execution semantics of templates imposing a directly-follows relation between activities (e.g., Chain Response(x,y)).
- A DPN is considered to be permanently violated if an event included in that DPN occurs while no corresponding transition can be successfully fired in its current state.
- A DPN is considered to be temporarily violated until the final marking of that DPN is reached or a permanent violation occurs. Furthermore, a DPN is considered permanently satisfied if its final marking is reached. This captures the intuition that a DPN specifies some procedural sequence of events to be carried out once during the full process control flow.
- The occurrence of each event is processed against all process specifications simultaneously (except the specifications ignoring that event). This captures the intuition of concurrent execution: if multiple specifications require the same event to occur, then its single occurrence should satisfy all of them.
- The variables of each DPN are local to that DPN. This keeps the semantic meaning of each DPN intact in the assimilated hybrid process specification.
- An LMP-Declare specification is considered permanently satisfied iff all individual constraints within that specification are permanently satisfied. This captures the intuition that an LMP-Declare specification represents declarative knowledge applicable throughout the full process control flow.
- An LMP-Declare specification is considered permanently violated iff at least one individual constraint within that specification is permanently violated. This captures the intuition that the constraints in a single LMP-Declare specification form a single logical entirety.

5.2 Monitoring Automaton

The proposed monitoring approach relies on a GFA, which is further annotated with violation and cost models, thus forming the hybrid process specification of the M3 Framework (Fig. 3). A GFA is a standard finite-state automaton, with the only difference that the transitions of the automaton are decorated with

data conditions, imposing additional restraints on when a specific transition is allowed. This representation allows for capturing the semantic meaning of each individual process specification (including the data perspective), while also fully accounting for the execution semantics outlined in Sect. 5.1. Additionally, the annotated monitoring automaton enables early conflict detection and next activity recommendations as outlined in Sect. 5.3.

The creation of the monitor automaton occurs at the end of the preparation phase of the M3 Framework (Sect. 4.2, Automated Assimilation) and consists of the following main steps:

Propositionalization — Translation of the original process specifications into propositionalized specifications based on interval abstraction. Activities and attribute-constant combinations (including intervals between the constants) used in the process specifications are extracted and enumerated to form propositions. The activity names and conditions are then replaced in the original process specifications with equivalent sets of propositions to form propositionalized specifications similarly to the approach presented in [3].

Individual GFA Construction — Creation of an equivalent GFA for each propositionalized process specification. LTL_f-based automata construction techniques are used for LMP-DECLARE and a reachability-based algorithm is used for DPNs. In case of DPNs, additional self loops are added to all states of the GFA for handling events that the DPN should ignore, and an additional trap state is added for handling permanent violations.

Cross-product GFA Construction — Combining the individual GFAs into a cross-product GFA. A standard automata cross-product algorithm can be used, given that the guards of the transitions are encoded as simple propositional strings.

Violation Model Integration — Annotation of the cross-product GFA states with a corresponding global monitoring status and a corresponding monitoring status of each individual process specification. In the current implementation, this is performed by concurrent traversal of the cross-product GFA and individual GFAs, however this can be further improved in the future.

Cost Model Integration — Fixpoint computation procedure based on the priorities of the input process specifications and the violation model from the previous step. The procedure begins by annotating each GFA state with the total cost of temporary and permanent violations incurred in that state. Here, the cost is equivalent to the priority of the specification, thus making the highest priority specifications also the most costly to violate. A copy of these annotations is made and then updated iteratively, based on the corresponding values of the successor states, so that the lowest value is used after each iteration. As a result, each state of the cross-product GFA is annotated with the total cost of stopping the process execution in the given state and the lowest possible total cost of violations achievable from the given state.

5.3 Event Processing

Event processing in the proposed monitoring approach is somewhat analogous to other existing automata based monitoring approaches [14]. The main differ-

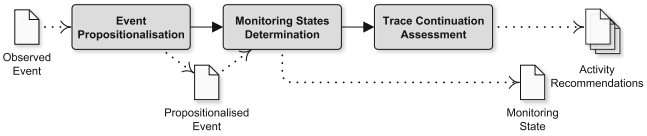


Fig. 4. Main steps and intermediate artifacts for processing a single event.

ences are that each incoming event must be translated into a propositionalized form accepted by the monitoring automaton, and two additional post-processing steps are performed after updating the state of the monitoring automaton, one for determining the monitor state, and one for determining the next activity recommendations. An overview of the event processing steps is provided on Fig. 4.

Processing of each observed event occurs at the beginning of the monitoring phase of the M3 Framework (Sect. 4.3, Automated Event Processing) and consists of the following main steps:

Event Propositionalization — Translation of the observed event into the equivalent propositionalized event. The propositionalized event is required to correspond to one of the propositions used in the monitoring automaton (Sect. 5.2, Propositionalization step) and can therefore be used directly to transition the monitor automaton into its corresponding next state.

Monitor State Determination — Determination of the monitor state based on the propositionalized event. The monitoring state can be looked up directly based on the corresponding GFA annotations and consists of a global monitoring state and the monitoring state of each individual process specification (Sect. 5.2, Violation Model Integration).

Trace Continuation Assessment — Determination of the next activity recommendations based on cost model annotations of the GFA. Each immediate successor state of the current monitor automaton state is evaluated to determine the set of states that can lead to a minimum total cost of violations. The propositional labels of the transitions leading to the best successor states are translated into equivalent events (including the data perspective) and returned as the set of activity recommendations.

6 Preliminary Experiments

A prototypical implementation of the monitoring approach presented in Sect. 5 was developed to perform preliminary experiments. In this section, we give an overview of these experiments by providing an example of the monitoring results and discussing scalability with respect to the number of input process specifications. For reproducibility purposes, the implemented tool, all models, and all event logs used in the experiments are available at https://git.io/JM0iA.

Monitoring Results. To demonstrate the monitoring results, we rely on the example scenario introduced in Sect. 2. We have modeled both the hip fracture

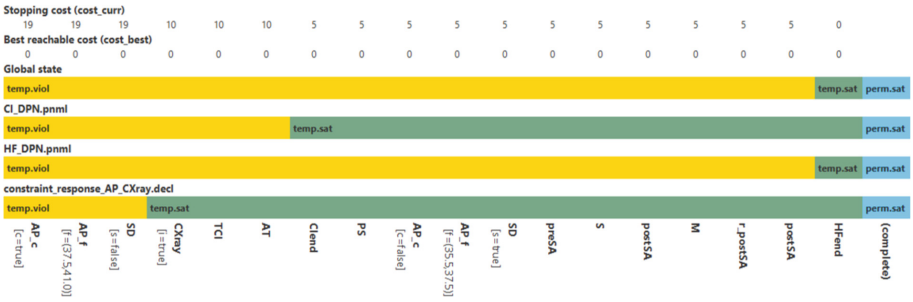


Fig. 5. Example of monitoring results (excluding activity recommendations) for a case where the patient is required to do a surgery and, at the same time, has a chest infection.

and the chest infection DPNs, as well as the constraint requiring an X-ray check if a patient has high body temperature or cough. The DPNs are assigned a violation cost of 5, while the constraint is assigned a violation cost of 9. This gives us a total of three process specifications, which, along with their violation costs, are automatically combined into a single hybrid process specification for monitoring.

A screenshot of the monitoring results, for a case where suspicion of a chest infection causes the surgery to be postponed, is provided in Fig. 5. There are a few key observations that we can make based on these monitoring results. First, the stopping cost drops after certain events, indicating that some process specifications have been completely and correctly carried out. Second, the best reachable cost is always zero, indicating that there is always at least one possible sequence of future events (including payloads) that would satisfy all process specifications. The last process specification (the declarative part of the hybrid specification) is temporarily violated after the first event because cough is detected, thus requiring a chest X-ray to be performed. While these are the main observations based on the given scenario, a more in-depth description, including an overview of the activity recommendations, is available at https://youtu.be/f9gy74Ikl8s.

Scalability. Multiple copies of the chest infection treatment DPN from Sect. 2 were used to asses the scalability of the monitoring approach. All activities in each copy were renamed to be distinct and a constraint such as Chain Response(`noCIcopy01`, `CXraycopy02`) was used to connect each new DPN copy to the previous one, therefore allowing to scale up the experiments in a predictable manner that would also include interplay of the process specifications. The same event log, consisting of four traces of three to four events, was used for all tests.

Based on the test results (Table 1), it is clear that, for the offline task of building the monitoring automaton, the scalability of the current implementation is reasonable when the number of input process specifications is not very large. While the memory usage and the size of the monitoring automaton are relatively

Table 1. Results of the syntetic scalability experiments.

Specifications		Monitoring Automaton		Event Processing(ms)			Implementation
DPN	LMP-Declare	Time(ms)	States	Min.	Max.	Mean	Memory(mb)
1	0	25	7	5	40	11	226
2	1	53	83	5	40	15	232
3	2	167	1010	4	45	15	269
4	3	1997	12092	5	47	15	829
5	4	18774	143272	4	49	16	8886

small for 3 DPN models with 2 LMP-Declare constraints, both requirements quickly ramp up with higher number of input specifications. Meanwhile, the performance of event processing is minimally affected by the number of input specifications and remained near instantaneous in all tests.

The potential scalability issues of the current implementation are mainly caused by two factors. First, we focused on demonstrating that multiple declarative and procedural process specifications can be combined for monitoring, thus advancing the state-of-the-art by overcoming the assumption that the process behavior should be embedded into a single process model. Given this focus, we opted for a naïve implementation that could be used as a baseline for future optimizations. Second, the approach itself has some inherent complexity issues due to propositionalization, which leads to a state explosion in the constructed automaton. We believe that these potential scalability issues can be mitigated; however, this requires further research.

7 Related Work

Monitoring Petri Net Models. There are various works that rely on different classes of Petri nets for capturing inappropriate system behaviors. One of the first works studies a monitoring approach that detects errors by controlling the number of tokens inside P-invariants [19]. Another work [18] proposes a workflow management system that encompasses workflow monitoring and delay prediction modules based on resource-aware Petri nets. While there are other Petri-net based monitoring approaches, all of them share the common assumption that the full process specification is given as a single Petri net, which can be insufficient in domains with highly flexible and knowledge-intensive processes.

Monitoring Declarative Specifications. Here, we restrict ourselves to approaches related to Declare (for comparisons with other approaches please refer to [12]). While existing Declare approaches in general assume that a single specification is used, splitting it into multiple specifications (e.g., one specification per constraint) would be semantically equivalent. This is well exemplified in [14], where the truth value of each constraint is monitored individually and possible global conflicts are handled by the conjunction of all the constraints being monitored. The Declare language has also been extended to account for

multiple perspectives by considering activity payloads and corresponding monitoring approaches have been developed (see, e.g.,[6,13]). However, none of these works studied monitoring of LMP-DECLARE constraints in hybrid specifications.

Conformance Checking. The interplay of multiple process specifications (both procedural and declarative) has however been addressed to some extent from the perspective of conformance checking. A recent work [8] studies the conformance checking task for mixed-paradigm process models that integrate Petri nets and DECLARE constraints. However, this setting does not consider any activity payloads and, as customary to conformance checking, the authors focus on alignment of complete traces and not on monitoring ongoing incomplete process executions. There are two other research lines that consider multiple process and constraint specifications, both related to the medical domain. The first one focuses on interactions between CGs and BMK from the view-point of the conformance checking problem [4]. The second one studies the same interactions, but from the perspective of explainability [22] (i.e., how can the actions taken during the treatment of a specific patient be automatically explained given the presence of multiple CGs and BMK). While these approaches consider the interplay of procedural and declarative models, their respective tasks of interest are performed on historical data and do not consider streams of events.

Hybrid Models. In addition to the aforementioned approaches, research in relation to hybrid business process representations (HBPRs) is currently ongoing. The term hybrid refers, in this case, to combining declarative and procedural modeling paradigms into a unified modeling approach which would allow expressing both strict and flexible aspects of a single process in the same model. A conceptual framework and a common terminology for these types of models has been proposed recently [2] and a number of open research challenges related to HBPRs have been identified [21]. Some process mining approaches for HBPRs [7,15,20] have also been developed. However, to the best of our knowledge, there are currently no monitoring approaches suitable for hybrid settings.

8 Conclusion

In this paper, we have presented M3, a framework for monitoring hybrid process specifications in domains where multiple interacting (procedural) sub-processes work under additional (declarative) constraints. We developed a prototypical implementation of the monitoring approach as a proof-of-concept and performed initial scalability experiments showing that, even without optimizations, the tool scales reasonably well when using a limited number of specifications.

The avenues for future work can be divided into three categories. First, the scalability of the approach should be improved, both by code optimizations and by exploring heuristics for reducing the monitoring automaton size. Second, the functionality should be extended to support the time perspective, recovery strategies, and runtime modifications of the hybrid specification. These would respectively allow the process analysts to monitor time critical aspects of a process (e.g., a mandatory checkup after a surgery within a certain time period),

continue monitoring after a permanent violation (thus enabling the detection of multiple violations of a single constraint), and handle unforeseen circumstances (e.g., a COVID-19 infection during an ongoing treatment or adaptations of a clinical guideline). Finally, while we have presented a general framework for utilizing our monitoring approach, there are additional socio-technical aspects that should be explored, e.g., how to integrate the framework into an organizational context, and what skills would be necessary to support its different phases.

Acknowledgements. The work of A. Alman was supported by the European Social Fund via "ICT programme" measure and by the Estonian Research Council grant PRG1226. F.M. Maggi was supported by the UNIBZ project CAT.

References

1. Alman, A., Maggi, F.M., Montali, M., Patrizi, F., Rivkin, A.: Monitoring hybrid process specifications with conflict management: The automata-theoretic approach. Technical report, arXiv.org (2021)
2. Andaloussi, A.A., Burattin, A., Slaats, T., Kindler, E., Weber, B.: On the declarative paradigm in hybrid business process representations: a conceptual framework and a systematic literature study. Inf. Syst. **91**, 101505 (2020)
3. Bergami, G., Maggi, F.M., Marrella, A., Montali, M.: Aligning data-aware declarative process models and event logs. In: BPM, pp. 235–251 (2021)
4. Bottrighi, A., Chesani, F., Mello, P., Montali, M., Montani, S., Terenziani, P.: Conformance checking of executed clinical guidelines in presence of basic medical knowledge. In: BPM Workshops, pp. 200–211 (2011)
5. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
6. De Masellis, R., Maggi, F.M., Montali, M.: Monitoring data-aware business constraints with finite state automata. In: ICSSP, pp. 134–143. ACM (2014)
7. De Smedt, J., De Weerdt, J., Vanthienen, J.: Fusion miner: process discovery for mixed-paradigm models. Decis. Support Syst. **77**, 123–136 (2015)
8. van Dongen, B.F., De Smedt, J., Di Ciccio, C., Mendling, J.: Conformance checking of mixed-paradigm process models. Inf. Syst. **102**, 101685 (2021)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, 3rd Edition. Addison-Wesley (2007)
10. Jalali, A., Maggi, F.M., Reijers, H.A.: A hybrid approach for aspect-oriented business process modeling. J. Softw. Evol. Process. **30**(8), e1931 (2018)
11. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. In: Trujillo, J.C., et al. (eds.) ER 2018. LNCS, vol. 11157, pp. 219–235. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00847-5_17
12. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015)
13. Maggi, F.M., Montali, M., Bhat, U.: Compliance monitoring of multi-perspective declarative process models. In: EDOC, pp. 151–160. IEEE (2019)
14. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: an approach based on colored

automata. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 132–147. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2_13
15. Maggi, F.M., Slaats, T., Reijers, H.A.: The automated discovery of hybrid processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 392–399. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_27
16. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2015). https://doi.org/10.1007/s00607-015-0441-1
17. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographiess. ACM Trans. Web **4**(1), 3:1–3:62 (2010)
18. Pla, A., Gay, P., Meléndez, J., López, B.: Petri net-based process monitoring: a workflow management system for process modelling and monitoring. J. Intell. Manuf. **25**(3), 539–554 (2012). https://doi.org/10.1007/s10845-012-0704-z
19. Prock, J.: A new technique for fault detection using petri nets. Automatica **27**(2), 239–245 (1991)
20. Sadiq, S.W., Orlowska, M.E., Sadiq, W.: Specification and validation of process constraints for flexible workflows. Inf. Syst. **30**(5), 349–378 (2005)
21. Slaats, T.: Declarative and hybrid process discovery: recent advances and open challenges. J. Data Semant. **9**(1), 3–20 (2020)
22. Spiotta, M., Terenziani, P., Theseider Dupré, D.: Temporal conformance analysis and explanation of clinical guidelines execution: an answer set programming approach. IEEE Trans. Knowl. Data Eng. **29**(11), 2567–2580 (2017)