RuM: Declarative Process Mining, Distilled

Anti Alman[1], Claudio Di Ciccio[2], Fabrizio Maria Maggi[3], Marco Montali[3], and Han van der Aa[4]

[1] University of Tartu, Tartu, Estonia, anti.alman@ut.ee
[2] Sapienza University of Rome, Rome, Italy, claudio.diciccio@uniroma1.it
[3] Free University of Bozen-Bolzano, Bolzano, Italy, {maggi,montali}@inf.unibz.it
[4] University of Mannheim, Mannheim, Germany, han@informatik.uni-mannheim.de

Abstract. Flexibility is a key characteristic of numerous business process management domains. In these domains, the paths to fulfil process goals may not be fully predetermined, but can strongly depend on dynamic decisions made based on the current circumstances of a case. A common example is the adaptation of a standard treatment process to the needs of a specific patient. However, high flexibility does not mean chaos: certain key process rules still delimit the execution space, such as rules that prohibit the joint administration of certain drugs in a treatment, due to dangerous interactions. A renowned means to handle flexibility by design is the declarative approach, which aims to define processes through their core behavioural rules, thus leaving room for dynamic adaptation. This declarative approach to both process modelling and mining involves a paradigm shift in process thinking and, therefore, the support of novel concepts and tools. Complementing our tutorial with the same name, this paper provides a high-level introduction to declarative process mining, including its operationalization through the RuM toolkit, key conceptual considerations, and an outlook for the future.

Keywords: Declare · Declarative Process Mining · Rule Mining · Process Discovery · Conformance Checking · Process Monitoring · Declarative Modelling

1 Introduction

Infusing flexibility in process-aware information systems is widely recognised as a key challenge in business process management (BPM) and information systems engineering [17]. Within the flexibility spectrum, *flexibility by design* advocates that process modelling languages themselves need to offer modelling primitives that provide freedom to process executors when deciding how to execute the process. The question then becomes how such languages help in finding a suitable trade-off between flexibility and control. Declarative approaches tackle this problem in an extreme way: the model indicates *what* the relevant temporal/dynamic constraints that have to be respected during process execution are, leaving the executors free to decide *how* to unfold the concrete executions.

After seminal papers on the topic were published between 1998 and 2003 within BPM and neighboring fields [5, 18, 19], the community started investigating declarative process modelling more systematically starting from 2006, when Pesic and van der Aalst proposed to apply temporal logic patterns [8] to declaratively capture process constraints [20], eventually leading to the DECLARE language and system [16] and to the definition

of a variety of reasoning tasks thanks to different logic-based formalisations [14, 15]. This interest was further fueled by the introduction of other declarative approaches, most prominently Dynamic Condition-Response Graphs [11].

A well-known issue with declarative approaches is that while they enjoy flexibility, they typically do not explicitly indicate how execution has to be controlled. In other words, conforming executions are only implicitly described as those that satisfy all the given constraints. Constraints, in turn, may be quite diverse from each other (e.g., indicating what *is expected* to occur, but also what should *not* happen). At the same time, constraints implicitly and mutually affect each other (a phenomenon referred to as *hidden dependencies* in the cognitive dimensions framework when evaluating the characteristics of notations [9, 15, 6]). This notoriously challenges understandability and interpretability of declarative process models [10], and calls for toolkits providing continuous support to the end users [9]. One such toolkit is RuM [2], which addresses some of the above-mentioned issues by providing a unified user interface for declarative process mining algorithms.

2 Declarative Process Mining with RuM

RuM [2] is the first software platform natively designed for declarative process modelling and process mining. RuM is based on the well-known modeling language DECLARE [16] and its multi-perspective extension MP-DECLARE [3], that is, DECLARE extended with data and time perspectives.

The following sections give a brief overview of RuM. Further information and the download link can be found in [2] and at `https://rulemining.org/`. To illustrate RuM's functionality, we use the Sepsis treatment process and event log described in [13].

Automated Process Discovery. RuM includes multiple algorithms for automatically discovering a process model. As usual for DECLARE, the discovered models consist of a set of *constraints* where each constraint describes one specific aspect of the process (i.e. constrains the behaviour of the process in a specific way).

In general, a constraint describes either the cardinality of an activity (i.e., the number of occurrence thereof in a trace) or a relation between two activities (i.e., how the occurrence of an activity requires or disables the occurrence of another one). For example, EXACTLY1(*ER Triage*) means that activity *ER Triage* will occur exactly once in each trace. CHAINRESPONSE(*ER Registration*, *ER Triage*) means that when activity *ER Registration* occurs then *ER Triage* will occur immediately after.

The discovered model is by default visualised using the standard DECLARE notation (Fig. 1). Alternatively, it is possible to represent the model procedurally as an equivalent deterministic finite state automaton (Fig. 2). Finally, it is possible to represent the entire model as a set of natural language sentences, e.g., "When *ER Registration* occurs, then *ER Triage* occurs immediately afterwards".

Conformance Checking and Monitoring. RuM provides two conformance checking approaches. The first one detects constraint fulfilments and violations (Fig. 3a), which pinpoint both the events that occur as specified in the model and those events that contradict it. The second approach is based on log alignments (Fig. 3b), as it identifies

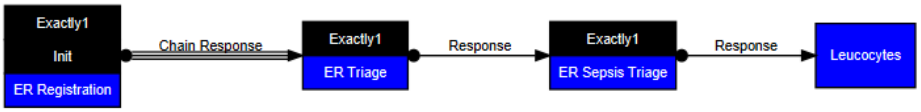


Fig. 1. An example of DECLARE model represented as a map.

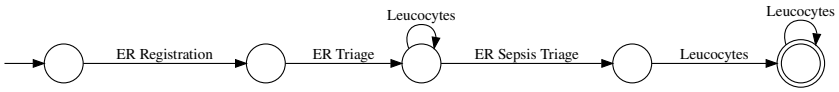
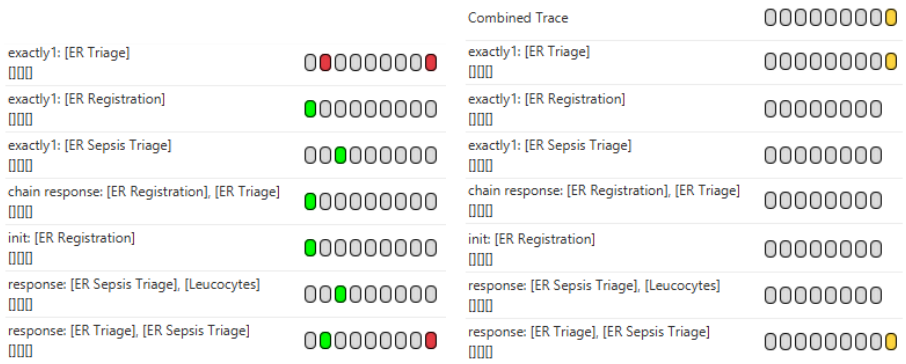


Fig. 2. An automaton representation of the DECLARE model in Fig. 1.



(a) Fulfilments (green) and violations (red) (b) Log alignment (deletions in yellow)

Fig. 3. Conformance checking approaches

the event insertions and/or deletions that would make the event log conforming with the DECLARE model.

The conformance checking results are provided at different levels, i.e., per event log, per trace, and per constraint. The latter is especially useful since it allows for clear insights and overall explainability of the conformance checking results with respect to specific process constraints. Additionally, RuM provides a monitoring functionality, which allows the user to interactively replay the traces one event at a time. During replay, RuM visualises the state of each constraint in the model (possibly/permanently satisfied and possibly/permanently violated) as the events of the trace are occurring.

Model Editing. RuM provides model editing capabilities through a fully MP-Declare compliant model editor, which supports the different representations discussed for process discovery above (cf., Figs. 1 and 2). These representations are updated and the inputs are validated on the fly as the model is being edited, thus directly showing how well a new or adapted constraint corresponds to the recorded process behaviour from an event log. In addition to editing the constraints directly, it is also possible to specify constraints and data conditions by using natural language speech and written text. This functionality is implemented as a simple chatbot named Declo [1].

Log Generation. Finally, RuM can generate event logs based on a given DECLARE model [7]. Although, by default, a generated log satisfies all constraints in the model, RuM also allows for the insertion of vacuous traces, i.e., traces that do not activate some constraints, and negative ones, i.e., traces that violate constraints. Together with other functions of RuM, this, for instance, supports the generation of event logs that match particularly important characteristics that were discovered from a real-world event log.

3 Considerations about Declarative Process Mining

The constraint-based nature of the declarative approach has various interesting implications and advantages with respect to the traditional, imperative paradigm. For instance, since declarative constraints establish behavioural rules that delimit the possible execution space for processes, they act like norms with which all process runs have to comply with. This characteristic make declarative models *open*, in that any execution is permitted as long as the expressed rules are not violated, as opposed to the *closed* scope of imperative models (e.g., Workflow nets, BPMN diagrams, event-process chains), which depict the whole execution space, from start to end [12].

From a mining perspective, declarative process mining aims at establishing, measuring and validating the rules that best *define* the behaviour emerging from the traces recorded in event logs – in the closest etymological sense of "defining", i.e., marking out their boundary. Therefore, exceptional, ad-hoc, or optional variants of process behaviour are fully supported as long as no constraints are expressed that contradict them. By contrast, an imperative model requires an alteration of its structure any time its unfolding does not encompass an alternative path that is evidenced in an event log. Declarative models can be used to represent the distinguishing core rules of event logs exposing high variability as per their stored runs, thereby catering for *flexibility*.

Declarative process rules are exerted over whole runs. Such a *global state perspective* differs from the imperative approach in which, given a current state, only the next enabled actions are made explicit (local state perspective). This reflects the difference between functional (declarative) and procedural (imperative) approaches to programming. A declarative rule applies any time a situation that triggers it is reached, regardless of the history of actions that led there. Also, the effect can span the whole execution of the process, i.e., at any moment in the future or the past: for instance, PRECEDENCE(*ER Sepsis Triage*, *IV Antibiotics*) requires that *ER Sepsis Triage* must occur at any point in time *before* the inoculation of antibiotics. On the contrary, imperative models dictate what the possible operations are for the next step given a case's history.

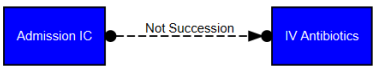


Fig. 4. A DECLARE negative constraint.

Declarative process models enjoy *compositionality* based on the conjunction of their constraints: the intersection of permitted behaviour from each rule determines the overall specification [4]. Adding rules restricts the range of acceptable runs. On the contrary, the addition of new states and transitions to an imperative model enlarges the execution space. We remark two consequences of this difference. Firstly, more flexible processes may create more cluttered imperative models (the so-called spaghetti

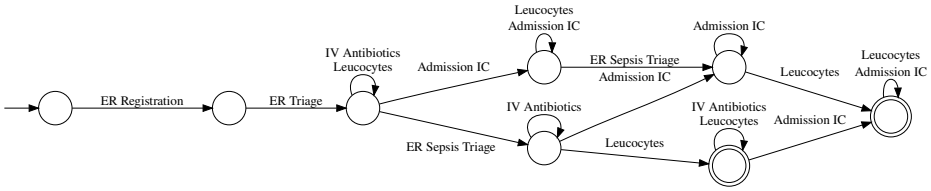


Fig. 5. The automaton representation of the model in Fig. 1 including the constraint in Fig. 4.

models) as declarative specifications would represent the core behavioural rules they are subject to rather than all the possible runs that would comply with them [16]. Secondly, declarative models are better suited for the seamless support of *negative* rules, i.e., constraints that impose the disablement of task occurrences given a specified condition. For example, NotSuccession(*Admission IC*, *IV Antibiotics*) imposes that after patients are admitted in the intensive care unit, they cannot undergo an inoculation of antibiotics. Adding this constraint to the model depicted in Fig. 1 is straightforward as declarative process models consist of lists of statements dictating the process rules. Graphically, it requires the sole juxtaposition of the constraint illustrated in Fig. 4 to the existing map. Including this constraint in the automaton representation in Fig. 2 requires the addition of numerous states and transitions as illustrated in Fig. 5 though.

To conclude, we remark that as the declarative process specifications dictate the rules that process executions are required to abide by, the space of runs that comply with it can be grounded in an imperative form [4]. Therefore, we claim that declarative process specifications can act as a bounding box within which imperative process models represent strategies to achieve the goals of the operating organisation, depending on the expertise, resources and context of the latter.

4 Research Opportunities

Beyond the current state of the art, we foresee several research opportunities in the context of declarative process modeling and mining. From a modelling perspective, a clear opportunity relates to the graphical notation. The original version of the Declare language is known to be difficult to understand [10] although it still lacks a means to depict relevant details such as resources or data. Although some effort has been done in this direction already, an extensive user evaluation to compare the different ways to represent constraints is still missing. Also, an interesting challenge is the analysis of declarative process models mixing crisp and probabilistic constraints, as discovered models often retain constraints that are violated by a set of traces in an event log. Another aspect currently under investigation pertains to the so-called *hybrid process models*, which consist of both imperative and declarative parts. This is important since real processes often contain both structured and unstructured parts. Finally, a highly promising research direction is the development of declarative modeling and mining instruments to deal with object-centric processes. This problem is also closely related to the assessment of the relevance of a constraint in a given process execution, which can depend on the nature of the actions and of the objects involved in the constraint.

Acknowledgements. The work of A. Alman was supported by the Estonian Research Council (project PRG1226) and ERDF via the IT Academy Program. The work of C. Di Ciccio was supported by MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science at Sapienza and by the Sapienza research project "SPECTRA".

References

1. A. Alman, K. J. Balder, F. M. Maggi, and H. van der Aa. Declo: A chatbot for user-friendly specification of declarative process models. In *BPM (PhD/Demos)*, pages 122–126, 2020.
2. A. Alman, C. Di Ciccio, D. Haas, F. M. Maggi, and A. Nolte. Rule mining with RuM. In *ICPM*, pages 121–128, 2020.
3. A. Burattin, F. M. Maggi, and A. Sperduti. Conformance checking based on multi-perspective declarative process models. *Expert Syst. Appl.*, 65:194–211, 2016.
4. 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.
5. 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.
6. J. De Smedt, J. De Weerdt, E. Serral, and J. Vanthienen. Discovering hidden dependencies in constraint-based declarative process models for improving understandability. *Inf. Syst.*, 74(Part 1):40–52, 2018.
7. C. Di Ciccio, M. L. Bernardi, M. Cimitile, and F. M. Maggi. Generating event logs through the simulation of declare models. In *EOMAS@CAiSE*, pages 20–36, 2015.
8. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE*, pages 411–420. ACM, 1999.
9. T. R. G. Green and M. Petre. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Vis. Comp. and Lang.*, 7(2):131–74, 1996.
10. 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.
11. 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.
12. 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 *CAiSE*, pages 270–285, 2012.
13. F. Mannhardt and D. Blinde. Analyzing the trajectories of patients with sepsis using process mining. In *RADAR+EMISA@CAiSE*, pages 72–80, 2017.
14. M. Montali. *Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach*, volume 56 of *LNBIP*. Springer, 2010.
15. M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative specification and verification of service choreographies. *ACM Trans. Web*, 4(1):3:1–3:62, 2010.
16. M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. DECLARE: Full support for loosely-structured processes. In *EDOC*, pages 287–300, 2007.
17. M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012.
18. 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.
19. M. P. Singh. Distributed enactment of multiagent workflows: Temporal logic for web service composition. In *AAMAS*, pages 907–914. ACM, 2003.
20. W. M. P. van der Aalst and M. Pesic. DecSerFlow: Towards a truly declarative service flow language. In *WS-FM*, pages 1–23, 2006.